

# Atmel SMART | SAM V7: Cortex-M7 Tutorial

## Using the SAMV7 Xplained ULTRA evaluation board

ARM Keil MDK 5 Toolkit Winter 2015 v 0.94 bob.boys@arm.com

**ARM KEIL**  
Microcontroller Tools

### Introduction:

The latest version of this document is here: [www.keil.com/appnotes/docs/apnt\\_274.asp](http://www.keil.com/appnotes/docs/apnt_274.asp)

The purpose of this lab is to introduce you to the Atmel Cortex<sup>®</sup>-M7 processor using the ARM<sup>®</sup> Keil<sup>®</sup> MDK toolkit featuring the IDE  $\mu$ Vision<sup>®</sup>. We will demonstrate all debugging features available on this processor including Serial Wire Viewer and ETM instruction trace. At the end of this tutorial, you will be able to confidently work with these processors and Keil MDK. We recommend you obtain the **new Getting Started MDK 5**: from here: [www.keil.com/mdk5/](http://www.keil.com/mdk5/).

**Keil Atmel Information Page:** See [www.keil.com/atmel/](http://www.keil.com/atmel/).

Keil MDK supports and has examples for most Atmel ARM processors. Check the Keil Device Database<sup>®</sup> on [www.keil.com/dd2](http://www.keil.com/dd2) for the complete list. Additional information is listed in [www.keil.com/Atmel/](http://www.keil.com/Atmel/).

**Linux:** Atmel ARM processors running Linux and Android are supported by ARM DS-5<sup>™</sup>. <http://www.arm.com/ds5>.

Keil MDK-Lite<sup>™</sup> is a free evaluation version that limits code size to 32 Kbytes. Nearly all Keil examples will compile within this 32K limit. The addition of a valid license number will turn it into a commercial version. MDK Atmel Edition is a one year renewable license for Atmel processors. Contact Keil Sales for details.

**Atmel 8051 Processors:** Keil has development tools for many Atmel 8051 processors. See [www.keil.com/Atmel/](http://www.keil.com/Atmel/) for details.

**Atmel | Start:**  $\mu$ Vision is compatible with the Atmel | START configuration program.

**RTX RTOS:** All variants of MDK contain the full version of RTX RTOS. RTX has a BSD license and source code is provided.  $\mu$ Vision provides two kernel awareness windows that update while the program is running. See [www.keil.com/rtx](http://www.keil.com/rtx).

Many other RTOSs are compatible with MDK as are applications with no OS (bare metal)

### Why Use Keil MDK ?

MDK provides these features particularly suited for Atmel Cortex-M users:

1.  $\mu$ Vision IDE with Integrated Debugger, Flash programmer and the ARM<sup>®</sup> Compiler/assembler/linker toolchain. GCC or LLVM can also be used. MDK is a complete turn-key "out-of-the-box" tool solution.
2. **Keil Middleware:** Network, USB, File System and Graphics. For more information see [www.keil.com/mdk5/middleware/](http://www.keil.com/mdk5/middleware/)
3. TÜV certified. SIL3 (IEC 61508) or ASILD (ISO 26262).
4. **Compiler Safety Certification Kit:** [www.keil.com/safety/](http://www.keil.com/safety/)
5. All applicable ARM CoreSight<sup>™</sup> debugging technology is supported.
6. Available Debug Adapters: Atmel EDBG (is CMSIS-DAP compliant), Keil ULINK<sup>™</sup>2, ULINK-ME, ULINK<sup>pro</sup> and Segger J-Link Ultra or Plus.
7. Keil Technical Support is included for one year and is easily renewable. This helps you complete your project faster and easier.
8. MDK includes board support packages for most Atmel evaluation boards.
9. Affordable perpetual and term licensing. Contact Keil sales or your local distributor for pricing options and special offers. See the last page.

**This document includes details on these features plus more:**

1. Real-time Read and Write to memory locations for the Watch, Memory and peripheral windows. These are non-intrusive to your program. No CPU cycles are stolen. No instrumentation code is added to your source files.
2. Six Hardware Breakpoints (can be set/unset on-the-fly) and two Watchpoints (also known as Access Breaks).
3. RTX and RTX Tasks windows: Two kernel awareness windows that update while your program is running.
4. A DSP example program using ARM CMSIS-DSP libraries and RTX.
5. ETM Instruction Trace including Performance Analyzer and Code Coverage.
6. How to create your own  $\mu$ Vision projects with and without RTX and a list of document resources available.



The SAMV71 Xplained Ultra board connected to a Keil ULINK<sup>pro</sup>. This lab uses the onboard Atmel EDBG.

## General Information:

1. CoreSight Definitions::	3
2. MDK 5 Keil Software Information:	4
3. Keil Software Download and Installation:	4
4. USB Debug Adapters:	4

## Software Packs:

5. $\mu$ Vision Software Packs Download and Install Process:	6
6. Examples Download and Install:	6
7. Other features of CMSIS-Pack Software Packs:	7

## Blinky Example and Debugging Features:

8. <i>Blinky</i> example using the Atmel SAMV71 Xplained and the EDBG adapter:	8
9. Hardware Breakpoints and Single Stepping:	9
10. Call Stack & Locals window:	10
11. Watch and Memory windows and how to use them:	11
12. System Viewer (SV): Peripheral Views:	12
13. Watchpoints: Conditional Breakpoints:	13
14. View Variables graphically using the Logic Analyzer:	14
15. printf using ITM:	15

## DSP Sine Example:

16. DSP Sine Example using ARM CMSIS-DSP Libraries and RTX	17
a) Running the DSP Example:	17
b) Serial Wire Viewer (SWV) and ETM Instruction Trace:	17
c) RTX System and Thread Viewer:	18
d) Event Viewer	19
e) Call Stack and Locals for the DSP Example:	20

## Keil Middleware:

17. Keil Middleware:	21
18. Middleware Examples: USB Mass Storage :	22

## Creating your own MDK 5 Blinky projects from scratch:

19. Creating your own MDK 5 Blinky project from scratch:	23
20. Creating your own MDK 5 RTX Blinky project from scratch:	26
21. Adding a Thread to your RTX Blinky:	27

## ETM Instruction Trace with ULINKpro:

22. ETM with ULINKpro:	28
23. Configuring ULINKpro ETM Trace:	28
24. Blinky with ETM Trace:	30
25. Code Coverage:	33
26. Performance Analysis:	35
27. Execution Profiling:	36
28. "In the weeds" Example:	37

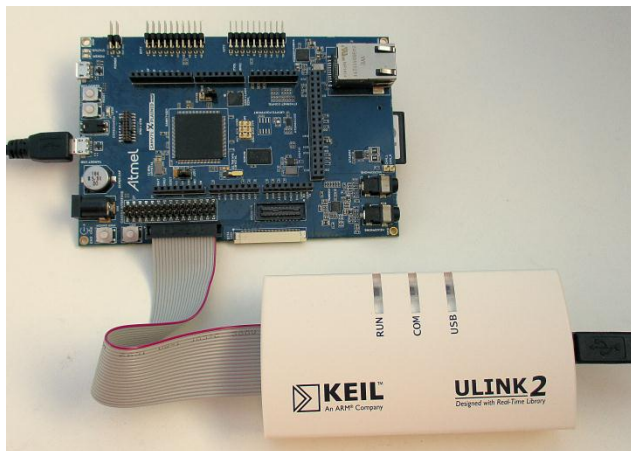
## Other Useful Information:

29. Serial Wire Viewer and ETM Summary:	38
30. Document Resources:	39
31. Keil Products and contact information:	40

## 1) CoreSight Definitions: *It is useful to have a basic understanding of these terms:*

Cortex-M0 and Cortex-M0+ often have features 2), 3), 11, 12 and sometimes 10) implemented. Cortex-M3, Cortex-M4 and Cortex-M7 have all features listed implemented except MTB. It is possible some processors have all features except ETM Instruction trace and the trace port. Consult your specific Atmel datasheet to determine its specific feature set.

1. **JTAG:** Provides access to the CoreSight debugging module located on the Cortex processor. It uses 4 to 5 pins.
2. **SWD:** Serial Wire Debug is a two pin alternative to JTAG and has about the same capabilities but no Boundary Scan. SWD is referenced as SW in the  $\mu$ Vision Cortex-M Target Driver Setup. Serial Wire Viewer (SWV) must use SWD because the JTAG signal TDIO shares the same pin as SWO. The SWV data normally comes out the 1 bit SWO pin.
3. **DAP:** Debug Access Port. This is a component of the ARM CoreSight debugging module that is accessed via the JTAG or SWD port. One of the features of DAP are the memory read and write accesses which provide on-the-fly memory accesses without the need for processor core intervention.  $\mu$ Vision uses DAP to update Memory, Watch, Peripheral and RTOS Thread Viewer windows in real-time while the processor is running. You can also modify variables on the fly. No CPU cycles are used, the program can be running and no source code stubs are needed. You do not need to configure or activate DAP.  $\mu$ Vision configures DAP when you select a function that uses it.
4. **SWV:** Serial Wire Viewer: A trace capability providing display of reads, writes, exceptions, PC Samples and printf.
5. **SWO:** Serial Wire Output: SWV frames usually come out this one pin output. It shares the JTAG signal TDIO.
6. **ITM:** Instrumentation Trace Macrocell: As used by  $\mu$ Vision, ITM is thirty-two 32 bit memory addresses (Port 0 through 31) that when written to, will be output on either the SWO or Trace Port. This is useful for printf type operations.  $\mu$ Vision uses Port 0 for printf and Port 31 for the RTOS Event Viewer. The data can be saved to a file.
7. **Trace Port:** A 4 bit port that ULINK $pro$  uses to collect ETM frames and optionally SWV (rather than SWO pin).
8. **ETM:** Embedded Trace Macrocell: Displays all the executed instructions. A ULINK $pro$  is needed for ETM. ETM requires a special 20 pin CoreSight connector. ETM also provides Code Coverage and Performance Analysis.
9. **ETB:** Embedded Trace Buffer: A small amount of internal RAM used as an ETM trace buffer. This trace does not need a specialized debug adapter such as a ULINK $pro$ . ETB runs as fast as the processor and is especially useful for very fast Cortex-A processors. Not all processors have ETB. See your specific Atmel datasheet.
10. **MTB:** Micro Trace Buffer. A portion of the device internal RAM is used for an instruction trace buffer. Only on certain Cortex-M0+ processors. Atmel Cortex-M3, Cortex-M4 and Cortex-M7 processors can provide ETM trace.
11. **Hardware Breakpoints:** The Atmel Cortex-M0+ has 2 breakpoints. The Atmel Cortex-M3, M4 and M74 have 6. These can be set/unset on-the-fly without stopping the processor. They are no skid: they do not execute the instruction they are set on when a match occurs.
12. **WatchPoints:** Both the Atmel Cortex-M0+, Cortex-M3, Cortex-M4 and Cortex-M7 have 2 Watchpoints. These are conditional breakpoints. They stop the program when a specified value is read and/or written to a specified address or variable. Check your datasheet for specific details on implementation. Some Cortex-M0/M0+ do not have data compare, only address compare.



Keil ULINK2



Segger J-Link Plus

## 2) MDK Keil Evaluation Software: MDK 5

**MDK 5** uses Software Packs to distribute processor specific software, examples and middleware. First you install MDK 5 Core. The Software Packs you require are then downloaded with the "Packs Installer", the version(s) selected with "Select Software Packs" and configured with the "Manage Run Time Environment" (RTE) utilities in  $\mu$ Vision. They can also be imported manually. You no longer need to wait for the next version of MDK or install patches to get the latest files.

**Atmel | START** provides software in MDK 5 format. MDK 4 projects are supported with a Legacy Pack.

MDK currently supports SAM E, S and V Atmel Cortex-M7 processors with Software Packs. Most other Atmel Cortex-M processors also have a Software Pack. See [www.keil.com/Atmel](http://www.keil.com/Atmel) for the current list.

**Keil Middleware:** Network, USB, File System and Graphics. For more information see [www.keil.com/mdk5/middleware/](http://www.keil.com/mdk5/middleware/). We recommend you obtain the latest Getting Started Guide for MDK5: It is available free of charge: [www.keil.com/mdk5/](http://www.keil.com/mdk5/).

**Keil software installation is a two step process as subsequently described:**

- Download and install the MDK Core file. This is done in Step 1 & 2 below.
- Download and install the appropriate Software Pack for the processor you are using. Details are on the next page.
- In addition, you need to download and install the examples used in this tutorial. These can be obtained from: [www.keil.com/apnotes/docs/apnt\\_274.asp](http://www.keil.com/apnotes/docs/apnt_274.asp) and from the Pack. Instructions are on page 6.

**Community Forums:** [www.keil.com/forum](http://www.keil.com/forum) and <http://community.arm.com/groups/tools/content>

## 3) Keil Software Download and Installation:

- Download MDK 5.17 or later from the Keil website. [www.keil.com/mdk5/install](http://www.keil.com/mdk5/install)
- Install MDK into the default folder. You can install into any folder, but this lab uses the default C:\Keil\_v5
- We recommend you use the default folders for this tutorial. We will use C:\00MDK\ for the examples.
- If you install MDK into a different folder, you will have to adjust for the folder location differences.
- You do not need a debug adapter: just the Atmel board, a USB cable and MDK installed on your PC.
- You do not need a Keil license for this tutorial. All examples in this tutorial will compile within the 32 K limit.
- You can obtain a one-time free 7 day license in File/License Management. If you are eligible, this button is visible: If you need additional time for evaluation purposes, please contact Keil sales. Contact information is on the last page of this tutorial.

Download MDK-Core Version 5

Evaluate MDK Professional

## 4) USB Debug Adapters:

**Keil manufactures several adapters.** These are listed below with a brief description. See [www.keil.com/ulink/](http://www.keil.com/ulink/)

- ULINK2:** ULINK2 supports Serial Wire Viewer (SWV). Run-time memory reads and writes for the Watch, Memory and Peripheral windows plus hardware breakpoints can be set/unset on-the-fly are all provided. See page 3.
- ULINKpro:** This is pictured on page 1. ULINKpro supports all SWV features and adds ETM Instruction Trace support. ETM records all executed instructions. ETM provides complete Code Coverage, Execution Profiling and Performance Analysis features. ULINKpro also provides the fastest Flash programming times. See page 1.

**Keil supports more adapters:**

- EDBG: (CMSIS-DAP):** An extra processor on your board becomes a debug adapter compliant to CMSIS-DAP. Atmel EDBG has a CMSIS-DAP mode which is selected in the  $\mu$ Vision Target Options menu under the Debug tab. EDBG currently does not support SWV or ETM. Operation is indicated by the yellow STATUS LED. OFF: no connection (not in Debug mode). ON: connected to target and program running. Blinking: If in Debug mode, program is stopped. If not in Debug mode, Flash is being programmed or other operation.
- Atmel-ICE:** Atmel ICE is also CMSIS-DAP compliant. It is selected as CMSIS-DAP in  $\mu$ Vision.
- SAM-ICE:** SAM-ICE (blue) is configured as a J-Link. Serial Wire Viewer is usable on V 6.0 and later. See page 6.
- Segger J-Link:** J-Link Version 6 (black) or older versions of J-Link Ultra do not support Cortex-M7. When you try to use such a J-Link  $\mu$ Vision will display a notice box. If a  $\mu$ Vision box offering to update the J-Link firmware is displayed, your J-Link is Cortex-M7 compatible once this firmware is upgraded. Most J-Links support Serial Wire Viewer. SWV data reads and writes are not displayed with a J-Link. ETM trace is not supported. See page 3.






## 5) µVision Software Pack Download and Install Process: (do 3 steps this page)

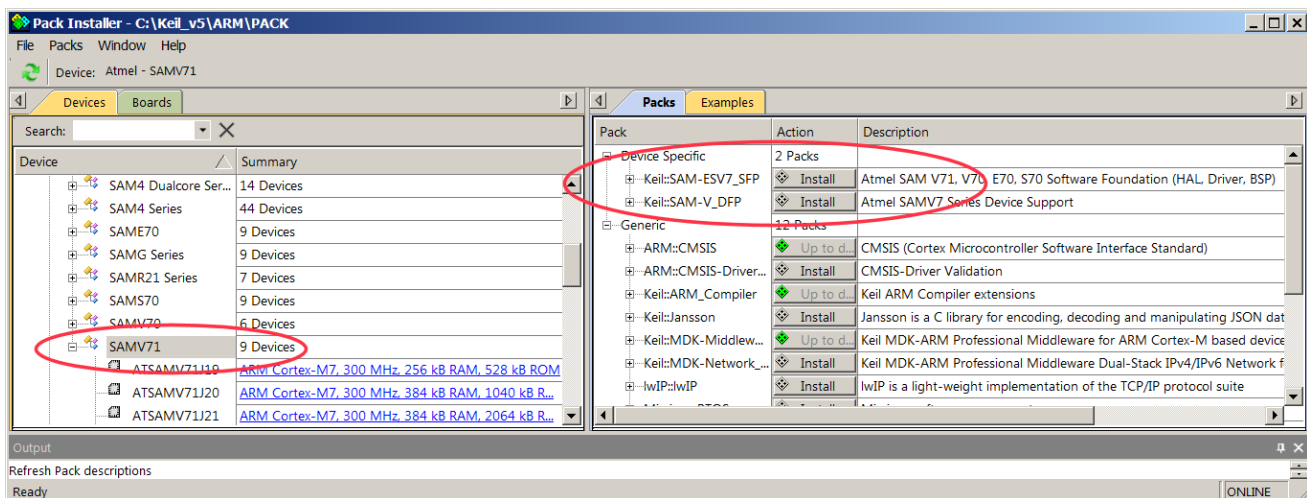
A Software Pack contains components such as header files, Flash programming, documents and other files used in a project. A Pack is a standard zip archive with a .pack file extension. It contains a .pdsc description file in XML format.

**TIP:** If you get an error when downloading a Pack: make sure your internet connection allows an application such as Pack Installer to download .zip files. A few do not. In this case, download your Pack from [www.keil.com/pack](http://www.keil.com/pack) and install it manually by double clicking on it. This is also an ideal method to share your own private Packs.

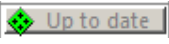
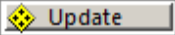
Information on creating your own Software Pack is found here: [www.keil.com/pack/doc/CMSIS/Pack/html/index.html](http://www.keil.com/pack/doc/CMSIS/Pack/html/index.html)

### 1) Start µVision and open Pack Installer (PI):

1. Connect your computer to the Internet. This is needed to download the Software Packs. Start µVision: 
  2. Open Pack Installer by clicking on its icon:  A Pack Installer Welcome screen will open. Read and close it.
  3. The Pack Installer window opens up as shown below:
  4. Note “ONLINE” is displayed at the bottom right of the Pack Installer window shown below: If “OFFLINE” is displayed, connect to the Internet before continuing.
  5. Select the Update icon  to refresh the Pack Installer listings. It is a good practice to do this regularly.
  6. Select the Devices tab on the left:
- TIP:** The left side (Devices and Boards) filters what is displayed on the right side (Packs and Examples).
7. Select Atmel and then SAMV71 as shown below:



### 2) Install the SAM V7 DFP Software Pack: (DFP = Device Family Pack)

1. Select Keil::SAM-V\_DFP under the Packs tab and click on Install. This Pack will download and install to C:\Keil\_v5\ARM\Pack\Keil\SAM-V\_DFP. This download can take several minutes.
2. Its status will be indicated by the “Up to date” icon: 
3. Update means there is an updated Software Pack available for download. 

**TIP:** You have the option of selecting which version of a Pack you want to use. See page 7.

### 3) Install the SAM ESV7 Software Pack: (SFP = Software Foundation Pack)

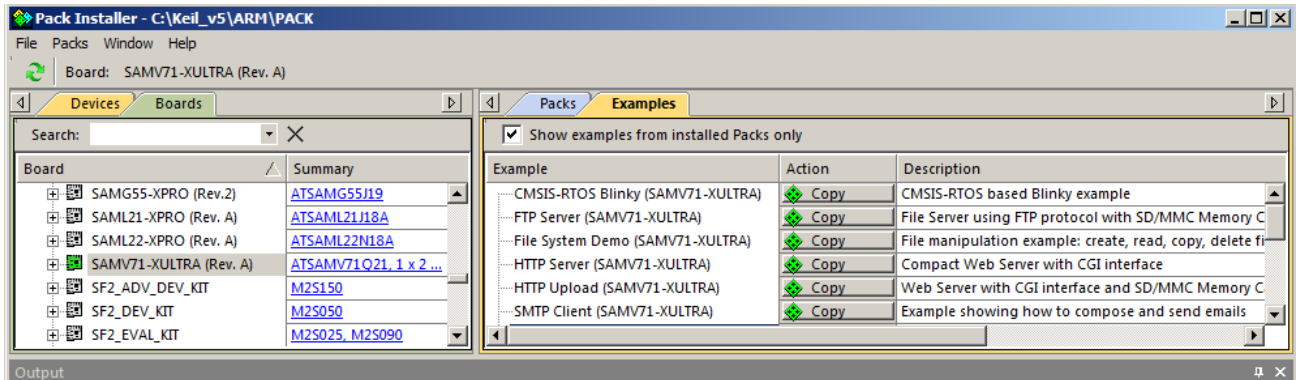
This Pack contains the Chip Library from the [Atmel Software Package](http://www.keil.com/pack/doc/SAM_ESV7/General/html/index.html) for the SAM-SAM-E7x, SAM-S7x, and SAM-V7x. Information regarding these Packs can be accessed here: [www.keil.com/pack/doc/SAM\\_ESV7/General/html/index.html](http://www.keil.com/pack/doc/SAM_ESV7/General/html/index.html)



1. Select Keil::SAM\_ES7\_SFP under the Packs tab and click on Install. This Pack will download to C:\Keil\_v5\ARM\Pack\Keil\SAM-ESV7\_SFP. You can look at these files but do not change anything.
2. Details about this SFP Pack can be accessed here: [www.keil.com/pack/doc/SAM\\_ESV7/General/html/index.html](http://www.keil.com/pack/doc/SAM_ESV7/General/html/index.html)

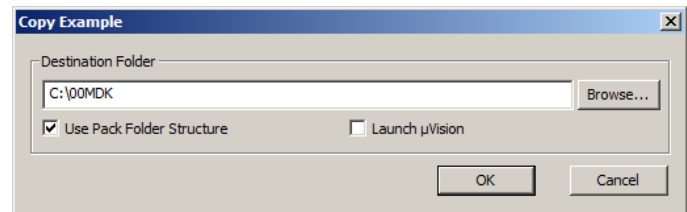
## 6) Examples Download and Install Process: (do 2 steps on this page)

### 1) Install the SAMV71 Xplained Ultra RTX\_Blinky Example:

1. Select the Boards tab. Select SAM V71 XULTRA.
2. Select the Examples tab. All the examples for this board are now filtered and visible.



3. Highlight CMSIS-RTOS Blinky (SAMV71-XULTRA) It is shown above under the Example column.
4. Select Copy  opposite CMSIS-RTOS Blinky (SAMV71-XULTRA).
5. The Copy Example window shown below opens up: Select Use Pack Folder Structure. Unselect Launch µVision.
6. Type in C:\00MDK. Click OK to copy the RTX\_Blinky project as shown below:
7. The CMSIS-Blinky example will now copy to C:\00MDK\Boards\Atmel\SAMV71-XULTRA. The folder created will be Blinky.
8. The rest of the path after C:\00MDK\ is calculated by Pack Installer.
9. We will download other examples later using the Pack Installer.
10. Close the Packs Installer. You can open it any time by clicking on its icon. 
11. If a window opens up saying "Software Pack folder has been modified": click on Yes.

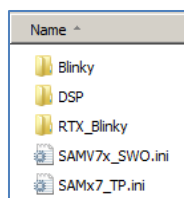


**TIP:** The default folder for copied examples the first time you install MDK is C:\Users\<user>\Documents. For simplicity, we will use the default folder of C:\00MDK\ in this tutorial. You can use any folder you prefer.

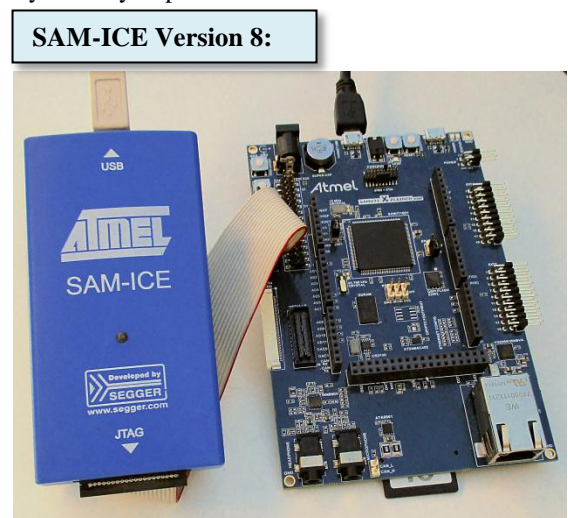
### 2) Install the Examples provided by this Tutorial:

The DSP and RTX\_Blinky examples are provided where you obtained this document: [www.keil.com/appnotes/docs/apnt\\_274.asp](http://www.keil.com/appnotes/docs/apnt_274.asp)

Extract them into C:\00MDK\Boards\Atmel\SAMV71-XULTRA. This folder will result:




The two .ini files shown will be used to configure the processor for SWV and ETM trace operations respectively. You will need them to add to your own projects that use SWV or ETM traces. SWO = Serial Wire Output and TP = Trace Port.

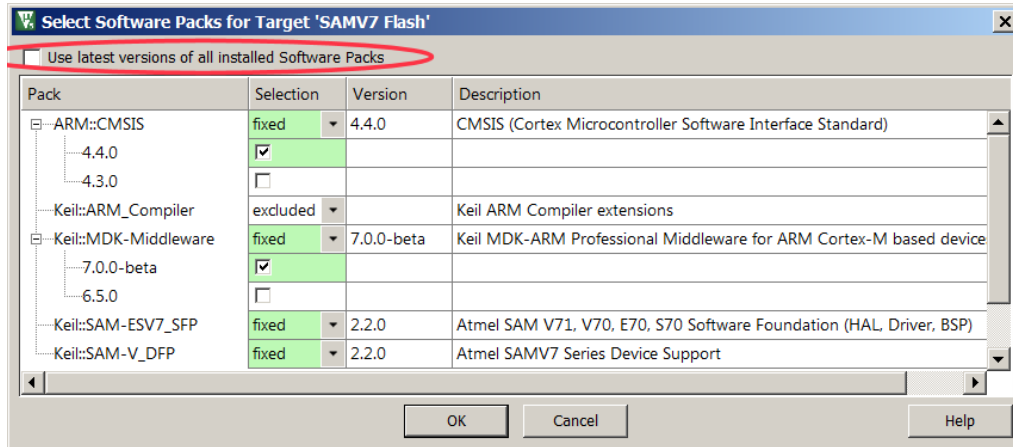


## 7) Other features of CMSIS-Pack Software Packs:

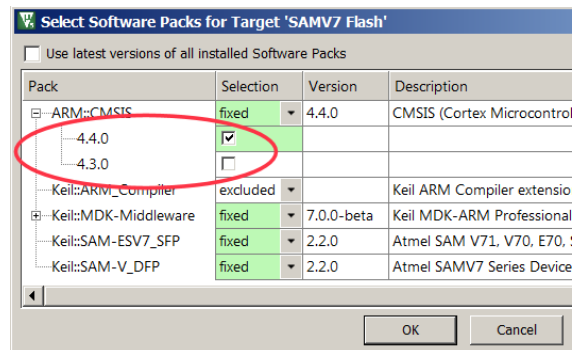
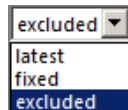
### A) Select Software Pack version:

This feature provides the ability to choose various Software Pack versions installed in your computer. You can select the versions you want to use. **TIP:** You can create several sets by creating a new target under Projects/Manage/Project Items...

1. Open the Select Software Packs by clicking on its icon: 
2. This window opens up. Note Use latest versions of all Installed ... is selected. The latest Packs will be used.
3. Unselect this setting and the window changes as shown below:




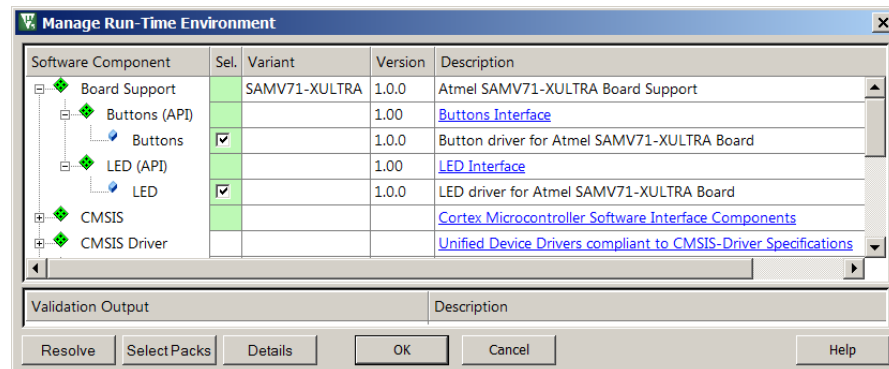
4. Expand the header ARM::CMSIS as shown above. Note various options are visible. You may see different versions depending on the Packs installed on your computer.
5. Select excluded and see the options as shown:
6. If you wanted to use V 4.3.0, you would select fixed then select the check box opposite 4.3.0.
7. Select Use latest...
8. Close this window.



and

### B) Manage Run-Time Environment:



1. Select the Blinky project in C:\00MDK\Boards\Atmel\SAMV71-XULTRA\Blinky\Blinky.uvprojx.
2. Click on the Manage RTE icon:  The next window opens:
3. Expand various headers and note the selections you can make. A selection made here will automatically insert the appropriate source files into your project when you click OK. (do not do this now !)
4. Do not make any changes at this time. Click Cancel to close this window.



## 8) *Blinky* example using the Atmel SAMV71 Xplained and the EDBG adapter:


We will connect the Keil MDK development system to the Xplained board using the on-board EDBG as the debug adapter. Atmel EDBG is CMSIS-DAP compliant.

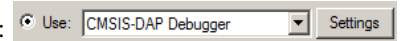
### Connection and Preparation:

1. Connect a USB cable between your PC and USB connector DEBUG USB.   
The green POWER LED will illuminate.
2. Start  $\mu$ Vision by clicking on its desktop icon. 
3. Select Project/Open Project.
4. Open the file: C:\00MDK\Boards\Atmel\SAMV71-XULTRA\Blinky\Blinky.uvprojx.



### Select the EDBG CMSIS-DAP Debugger:

5. Select Target Options  or ALT-F7 and select the Debug tab. Select CMSIS-DAP:
6. Select Settings: on the right side of this window. Confirm there is a valid IDCODE:






IDCODE	Device Name
0x0BD11477	ARM CoreSight SW-DP

If nothing or an error is displayed in the SW Device box, this *must* be corrected before you can continue. Check your USB connections. To refresh: switch to JTAG in Port: box and back to SW. It might take some time for the EDBG drivers to install.

7. Click OK twice to return to the main  $\mu$ Vision menu.

### Compile and RUN the Blinky Program:

8. Compile the source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
9. Enter Debug mode by clicking on the Debug icon.  The Flash memory will be programmed. Progress will be indicated in the Output Window. Select OK if the Evaluation Mode box appears. The yellow LED STATUS will blink a few times and then stay on.
1. Click on the RUN icon.  The yellow LED STATUS will blink indicating the program is running.

**Note:** You stop the program with the STOP icon. 

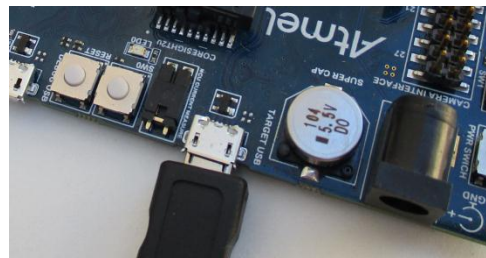
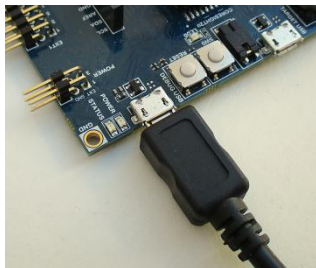
***LEDo and LED1 will now blink alternatively on the Atmel Xplained board.***

**Now you know how to compile a program, program it into the Atmel processor Flash, run it and stop it !**

**Note:** The board will start Blinky stand-alone. Blinky is now permanently programmed in the Flash until reprogrammed. Each LED is on for 500 msec and off for 1.5 sec. This is mostly due to the two `osDelay(500)` function calls in Blinky.c..

**TIP:** To use the on-board EDBG debugger, connect a USB cable to the USB connector DEBUG USB to power the board. To use an external debugger such as a Keil ULINK2, ULINK $pro$ , SAM-ICE or a J-Link Plus or Ultra +, connect USB power to the connector labelled TARGET USB or to the barrel power connector labelled VIN.

**TIP:** When using an external debugger and the green POWER LED blinks and a USB cable connected to TARGET USB connector, it is possible the board is using more power than your PC USB port can provide. Use a 5 volt external supply connected to VIN or a stronger USB source. You can try using DEBUG USB but risk conflict with the EDBG.






## 9) Hardware Breakpoints and Single Stepping:

The Atmel SAM V7 has six hardware breakpoints that can be set or unset on the fly while the program is running. This feature works using a CMSIS-DAP compatible such as EDBG or any Keil ULINK or a J-Link Plus or Ultra debug adapter.

1. With Blinky running, in the Blinky.c window, click on a darker grey block on the left on a suitable part of the source code. This means assembly instructions are present at these points. Inside the while (1) loop inside the main() function between near lines 42 through 55 is a good place: You can also click in the Disassembly window on a similar block to set a breakpoint.
2. A red circle will appear and the program will presently stop.
3. Note the breakpoint is displayed in both the Disassembly and source windows as shown here:
4. Set a second breakpoint in the while() loop as before.

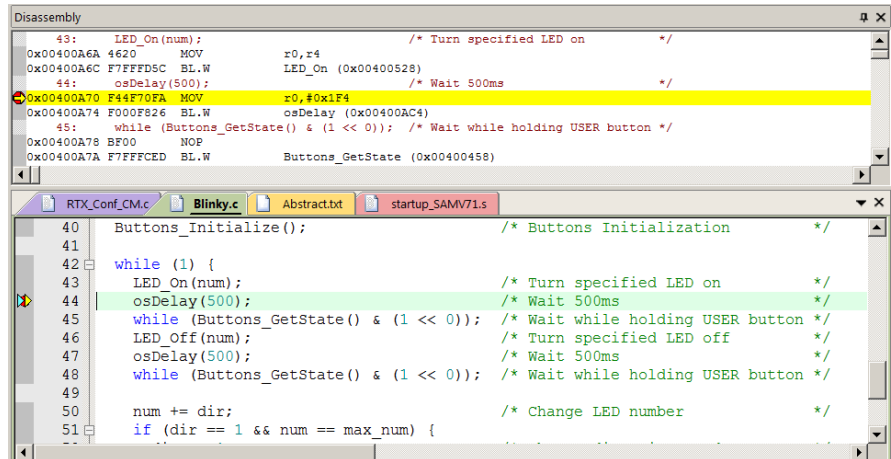
5. Every time you click on the

RUN icon  the program will run until the breakpoint is again encountered.

6. The yellow arrow is the current program counter value.

7. Clicking in the source window will indicate the appropriate code line in the Disassembly window and vice versa. This is relationship indicated by the cyan arrow and the yellow highlight:

8. Open Debug/Breakpoints or Ctrl-B and you can see any breakpoints set. You can temporarily unselect them or delete them. This window also will contain any Watchpoints.




9. Delete all breakpoints and close the Breakpoint window.

**TIP:** If you set too many breakpoints,  $\mu$ Vision will warn you. Cortex-M0 usually has only two. Other Atmel Cortex-M processors usually have six.

**TIP:** ARM hardware breakpoints do *not* execute the instruction they are set to and land on. This will be the next instruction executed. CoreSight hardware breakpoints are no-skid. This is a rather important feature for effective debugging.

### Single-Stepping:

1. With Blinky.c in focus (Blinky.c tab is underlined), click on the Step In icon  or F11 a few times: You will see the program counter jumps one C line at a time. The yellow arrow indicates the next C line or instruction to be executed.

**TIP:** Step might not seem to do anything if the program is in the RTX idle daemon (this is likely in this example). The CPU is executing a Branch to itself instruction each time you click Step Into.

Set a breakpoint on one of the function calls osDelay(500); in Blinky.c. These are near lines 44 and 47.

The program will then stop outside of the idle daemon and Step will now function normally.


2. Click on the top margin of the Disassembly window to bring it into focus. Clicking Step Into now jumps the program counter one assembly instruction at a time.


## 10) Call Stack + Locals Window:

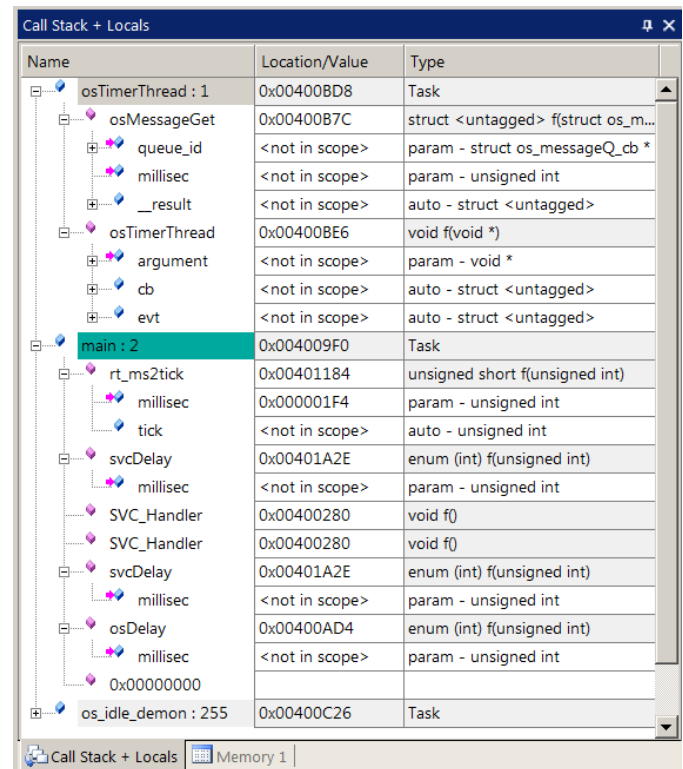
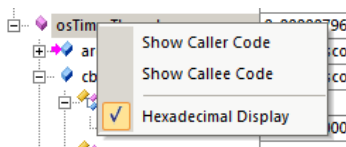
### Local Variables:

The Call Stack and Locals windows are incorporated into one integrated window. Whenever the program is stopped, the Call Stack + Locals window will display call stack contents as well as any local variables located in the active function.

If possible, the values of the local variables will be displayed and if not the message <not in scope> will be displayed. The Call + Stack window presence or visibility can be toggled by selecting View/Call Stack Window in the main µVision window when in Debug mode.

1. Set a breakpoint on one of the function calls to `osDelay(500);` in `Blinky.c`. These are located near lines 44 and 47.
2. Click on RUN .
3. Click on the Call Stack + Locals tab if necessary to open it. Expand some of the entries.
4. Shown is this Call Stack + Locals window:

5. The functions as they were called are displayed. If these functions had local variables, they would be displayed.
6. Click on the Step In icon  or F11 a number of times:
7. As you click on Step In, you can see the program entering and leaving various functions. Note the local variables are displayed.
8. If you get stuck in a delay or the `os_idle_Demon`, click on RUN to start over.
9. Note that this program has RTX RTOS running but there are no threads being switched as only thread was created. `main()` is the only thread created.
10. Right click on a function in the Call Stack and Locals window and select either Callee or Caller code and this will be highlighted in the source and disassembly windows.



11. When you ready to continue, remove the hardware breakpoint by clicking on its red circle ! You can also type Ctrl-B, select Kill All and then Close.

**TIP:** You can modify a variable value in the Call Stack & Locals window when the program is stopped.

**TIP:** This window is only valid when the processor is halted. It does not update while the program is running. Any local variable values are visible only when they are in scope.



**Do not forget to remove any hardware breakpoints before continuing.**

## 11) Watch and Memory Windows and how to use them:

The Watch and Memory windows will display updated variable values in real-time. It does this using the ARM CoreSight debugging technology that is part of Cortex-M processors. It is also possible to “put” or insert values into the Memory window in real-time. It is possible to “drag and drop” variable names into windows or enter them manually. You can also right click on a variable and select Add *varname* to.. and select the appropriate window. The System Viewer windows (peripheral views) and the System and Thread Viewer (RTX viewer) also use the same CoreSight technology.

### A) Watch window:

**Add a global variable:** Call Stack, Watch and Memory windows can’t see local variables unless stopped in their function.

1. Stop the processor  and exit Debug mode. 
2. Declare a global variable called **counter** near line 19 in Blinky.c:

**unsigned int counter = 0;**



3. Add these statements after LED\_On(num); near line 44:

**counter++;**

**if (counter > 0x0F) counter = 0;**

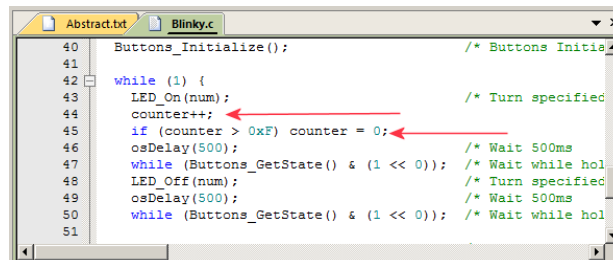
4. Select File/Save All. 

5. Click on Rebuild .

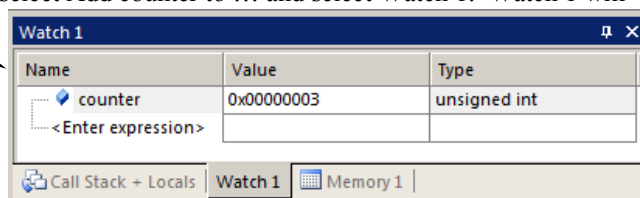
6. Enter Debug mode.  The Flash will be programmed. Click on RUN . You can configure a Watch window while the program is running. You can also do this with a Memory window.

7. In Blinky.c, right click on an instance of **counter** and select Add counter to ... and select Watch 1. Watch 1 will automatically open. **counter** will be displayed:

8. **counter** will update in real time.



```
40 Buttons_Initialize();
41
42 while (1) {
43     LED_On(num);
44     counter++;
45     if (counter > 0x0F) counter = 0;
46     osDelay(500);
47     while (Buttons_GetState() & (1 << 0));
48     LED_Off(num);
49     osDelay(500);
50     while (Buttons_GetState() & (1 << 0));
51 }
```



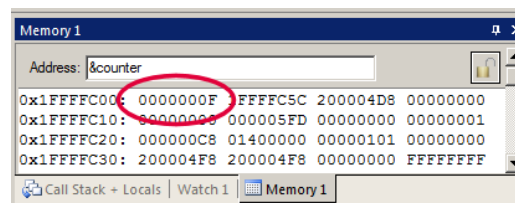
Name	Value	Type
counter	0x00000003	unsigned int
<Enter expression>		

**TIP:** To Drag ‘n Drop into a tab that is not active, pick up the variable and hold it over the tab you want to open; when it opens, move your mouse into the window and release the variable.

**TIP:** A Watch or Memory window can display and update global and static variables, structures and memory and peripheral addresses while the program is running. These are unable to display local variables because these are typically stored in a CPU register. These cannot be read by µVision in real-time. To view a local variable in these windows, convert it to a static or global variable.

### B) Memory window:

1. Right click on **counter** and select Add counter to ... and select the Memory 1 window.
2. Note the value of **counter** is displaying its address in Memory 1 as if it is a pointer. This is useful to see what address a pointer is pointing to but this not what we want to see at this time.
3. Add an ampersand “&” in front of the variable name and press Enter. The physical address is 0x2040\_0000.
4. Right click in the memory window and select Unsigned/Int.
5. The data contents of **counter** is displayed as shown here:
6. Both Watch and Memory windows are updated in real-time.
7. Right-click with the mouse cursor over the desired data field and select Modify Memory. You can change a memory location or variable on-the-fly while the program is still running.



Address	&counter
0x1FFFC000	0000000F FFFFC000 200004D8 00000000
0x1FFFC010	00000000 000005FD 00000000 00000001
0x1FFFC020	000000C8 01400000 00000101 00000000
0x1FFFC030	200004F8 200004F8 00000000 FFFFFFFF

**TIP:** No CPU cycles are used to perform these operations.

**TIP:** To view variables and their locations use the Symbol window. Select View/Symbol Window while in Debug mode.

## 12) System Viewer (SV):

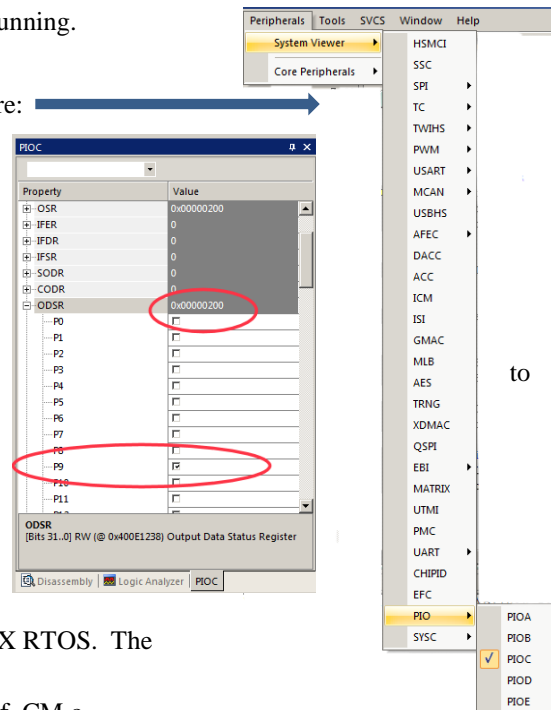
The System Viewer provides the ability to view certain registers in the CPU core and in peripherals. In most cases, these Views are updated in real-time while your program is running. These Views are available only while in Debug mode. There are two ways to access these Views: **a) View/System Viewer** and **b) Peripherals/System Viewer**. In the Peripheral/Viewer menu, the Core Peripherals are also available:

In our Blinky example, LED0 is connected to Port A pin 23 (PA23) and LED1 is connected to Port C pin 9 (PC09).

1. Click on RUN. You can open SV windows when your program is running.

### GPIO Port C:



2. Select Peripherals/System Viewer, PIO and then PIOC as shown here:
3. This window opens up. Expand ODSR:
4. You can now see P9 update as the LEDs blink in succession:
5. You can change the values in the System Viewer on-the-fly. When LED1 is off, click in the P9 box and it will come on.
6. This window is updated using the same CoreSight DAP process as the Watch and Memory windows.
7. Look at other Peripherals contained in the System View windows see what else is available.



**TIP:** If you click on a register in the properties column, a description about this register will appear at the bottom of the window.

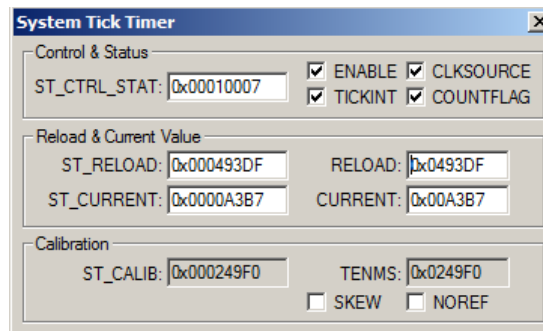
**SysTick Timer:** This program uses the SysTick timer as a tick timer for RTX RTOS. The function `osDelay(500);` is used to slow the program down.

RTX has configured the SysTick timer using values contained in `RTX_Conf_CM.c`.

1. Select Peripherals/Core Peripherals and then select SysTick Timer. Run the program.
2. The SysTick window shown below opens:
3. Note it also updates in real-time while your program runs using CoreSight DAP technology.
4. Note the `ST_RELOAD` and `RELOAD` registers. This is the reload register value. This is set during the SysTick configuration by RTX using values set in `RTX_Conf_CM.c`.
5. Note that it is set to `0x493DF`. This is created by  $300 \text{ MHz} / 0x493DF = 0x3E8 = 1,000$  which is the value in `RTX_Conf_CM.c`. Changing the reload value changes how often the SysTick timer creates its interrupt 15.
6. In the `RELOAD` register in the SysTick window, *while the program is running*, type in `0x7000` and click inside `ST_RELOAD` ! (or the other way around)
7. The blinking LEDs will speed up. This will convince you of the power of ARM CoreSight debugging.
8. Replace `RELOAD` with `0x493DF`. A CPU RESET  will also accomplish this.
9. When you are done, stop the program  and close all the System Viewer windows that are open.

**TIP:** It is true: you can modify values in the SV while the program is running. This is very useful for making slight timing value changes instead of the usual modify, compile, program, run cycle.


You must make sure a given peripheral register allows and will properly react to such a change. Changing such values indiscriminately is a good way to cause serious and difficult to find problems.



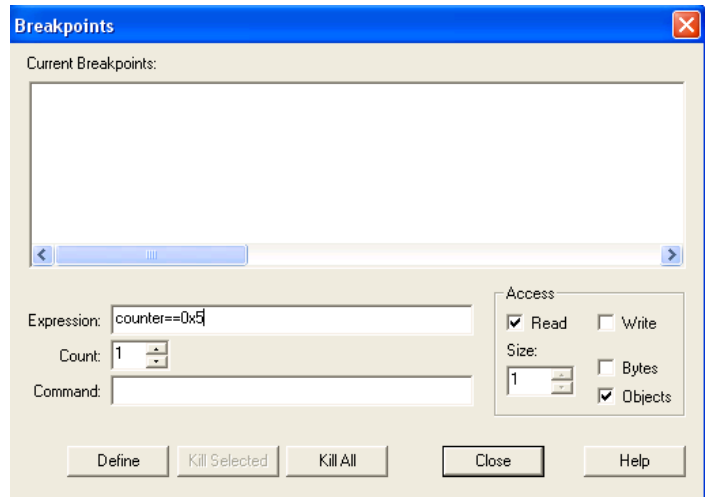



### 13) Watchpoints: Conditional Breakpoints

The Atmel Cortex processors have two Watchpoints. Watchpoints can be thought of as conditional breakpoints. Watchpoints are also referred to as Access Breaks in Keil documents. Cortex-M0+ Watchpoints are slightly intrusive. When the Watchpoint is hit, µVision must test the memory location. Cortex-M3/M4/M7 Watchpoints equality tests are not intrusive.

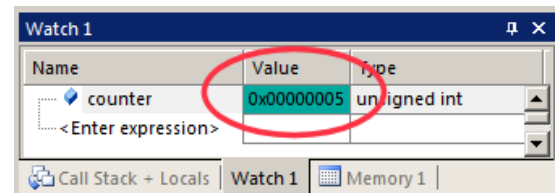
1. Use the same Blinky configuration as the previous page. Stop the program . Stay in Debug mode.
2. We will use the global variable **counter** you created in Blinky.c to explore Watchpoints.
3. Select Debug in the main µVision window and then select Breakpoints or press Ctrl-B.


4. Select Access to Read.
5. In the Expression box enter: “**counter == 0x5**” without the quotes. This window will display:
6. Click on Define or press Enter and the expression will be accepted as shown below in the bottom Breakpoints window:
7. Click on Close.
8. Enter the variable **counter** in Watch 1 if it is not already there.



9. Click on RUN. .
10. When **counter** equals 0x5, the Watchpoint will stop the program. See Watch 1 shown below:
11. Watch expressions you can enter are detailed in the Help button in the Breakpoints window. Triggering on a data read or write is most common. You can leave out the variable value and trigger on any Read and/or Write.
12. To repeat this exercise, click on RUN again. If counter does not get past 0x05, change **counter** to something other than 0x05 in the Watch window and click on RUN.

13. Stop the CPU. .
14. Select Debug/Breakpoints (or Ctrl-B) and delete the Watchpoint with Kill All and select Close.



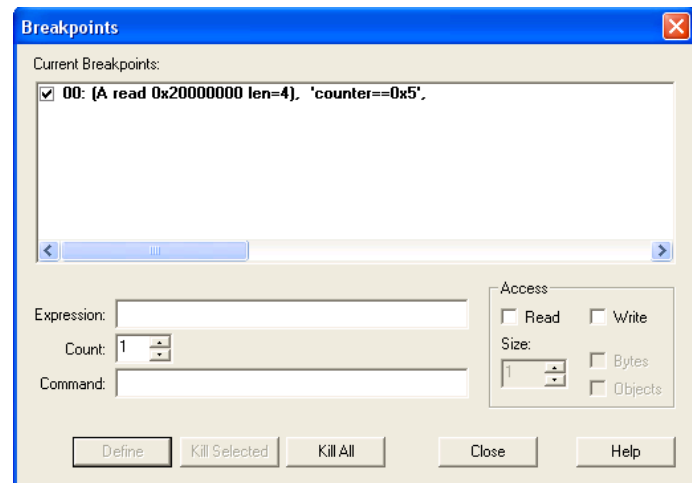
15. Exit Debug mode. .

**TIP:** You cannot set/unset Watchpoints on-the-fly while the program is running like you can with hardware breakpoints.

**TIP:** To edit a Watchpoint, double-click on it in the Breakpoints window and its information will be dropped down into the configuration area. Clicking on Define will create another Watchpoint. You should delete the old one by highlighting it and click on Kill Selected or try the next TIP:

**TIP:** The checkbox beside the expression allows you to temporarily unselect or disable a Watchpoint without deleting it.



**TIP:** Raw addresses can be used with a Watchpoint. An example is: \*((unsigned long \*) 0x20000004)



## 14) View Variables Graphically with the Logic Analyzer (LA):

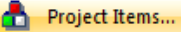








**This feature is only available with a Keil ULINK2, ULINKpro or a J-Link. EDBG does not currently support SWV.**

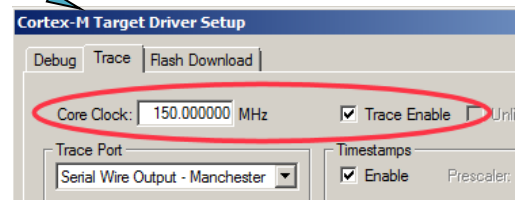
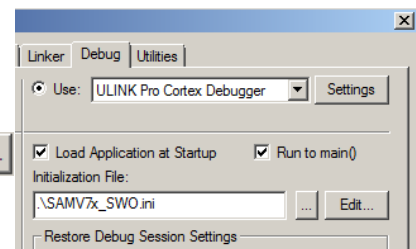
We will display the global variable counter you created earlier graphically in the Logic Analyzer.

1. Stop the processor  and exit Debug mode. 
2. Connect a ULINK2, ULINKpro, SAM-ICE or J-Link to the 20 pin SAMV71 DEBUG (SWD) as shown on page 3. Connect a ULINKpro to the 20 pin connector CORESIGHT SWD and ETM.
3. Connect a USB cable to connector DEBUG USB to provide 5 volt power.

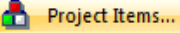







### A) ULINKpro: Configure Serial Wire Viewer (SWV):

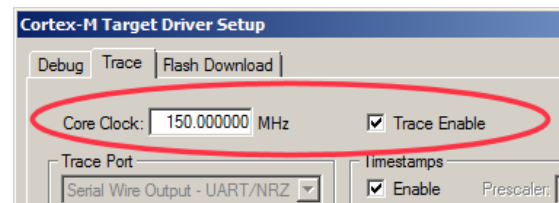
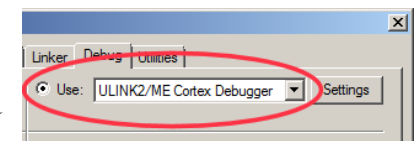
**TIP:** Using a ULINKpro, you can send SWV data either out the 1 bit SWO pin with Manchester encoding or out the 4 bit Trace Port. If sent out the Trace Port, you will need to slow the CPU speed down to 100 MHz or less. The Trace Port has a much greater throughput than the SWO pin. We will use the SWO pin in Manchester format in this tutorial.

4. **Create a new Target Options configuration:** Select Project/Manage/Project Items: 
5. Click on the Insert icon:  Enter **SAMV7 Flash SWV** (or whatever you want). Enter and then Close.
6. Select the Target Option you just created:  SAMV7 Flash SWV 
7. Select Target Options  or ALT-F7. Select the Debug tab. Select ULINK Pro Cortex Debugger as shown here: 
8. In the Initialization File: box, select SAMV7x\_SWO.ini using Browse icon 
9. Select Settings: on the right side of this window. Confirm SW is selected.
10. Select the Trace tab. In the Trace tab, select Trace Enable. Set Core Clock: to 150 MHz. (1/2 of 300 MHz)  
Set Trace Port to Serial Wire Output Manchester as shown here: 
11. Click OK twice to return to the main µVision menu.
12. Select File/Save All. 




### B) ULINK2: Configure Serial Wire Viewer (SWV):

1. **Create a new Target Options configuration:** Select Project/Manage/Project Items: 
2. Click on the Insert icon:  Enter **SAMV7 Flash SWV** (or whatever you want). Enter and then Close.
3. Select the Target Option you just created:  SAMV7 Flash SWV 
4. Select Target Options . Select the Debug tab.
5. Select Settings: on the right side of this window.
6. Select ULINK2/ME Cortex Debugger as shown here: 
7. In the Initialization File box, select SAMV7x\_SWO.ini using the Browse button . This file is provided with the Blinky example. It configures the processor CoreSight for trace operation. It is run when Debug mode is entered.
8. Select Settings: on the right side of this window.
9. Confirm SW is selected.
10. Select the Trace tab. In the Trace tab, select Trace Enable. Set Core Clock: to 150 MHz as shown here: (1/2 of 300 MHz)
11. Click OK twice to return to the main µVision menu.
12. Select File/Save All. 






**C) SAM-ICE and J-Link:** J-Link or SAM-ICE configuration is similar to ULINK2. Select J-Link J-Trace in the Use: selector.

**TIP:** If you get stuck here: you probably forgot to include SAMV7x\_SWO.ini:

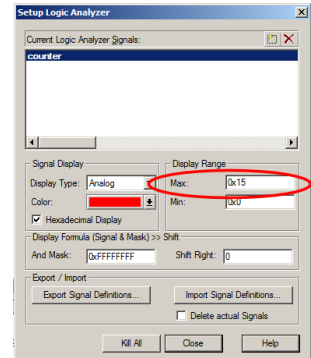
```
 164 while (ITM_PORT31_U32 == 0U);
```

## Configure the Logic Analyzer:

1. Enter Debug mode.  Click on Run. 
2. Open View/Analysis Windows and select Logic Analyzer or select the LA window on the toolbar. 

**TIP:** You can configure the LA while the program is running or stopped.

3. Click on the Blinky.c tab. Right click on **counter** and select Add 'counter' to... and then select Logic Analyzer. You can also Drag and Drop or enter it manually.
4. In the Logic Analyzer window, click on the Select box and the LA Setup window appears as shown here:
5. With `counter` selected, set Display Range Max: to 0x15 as shown here:
6. Click on Close.





**Run the Program: Note:** The LA can be configured while the program is running.

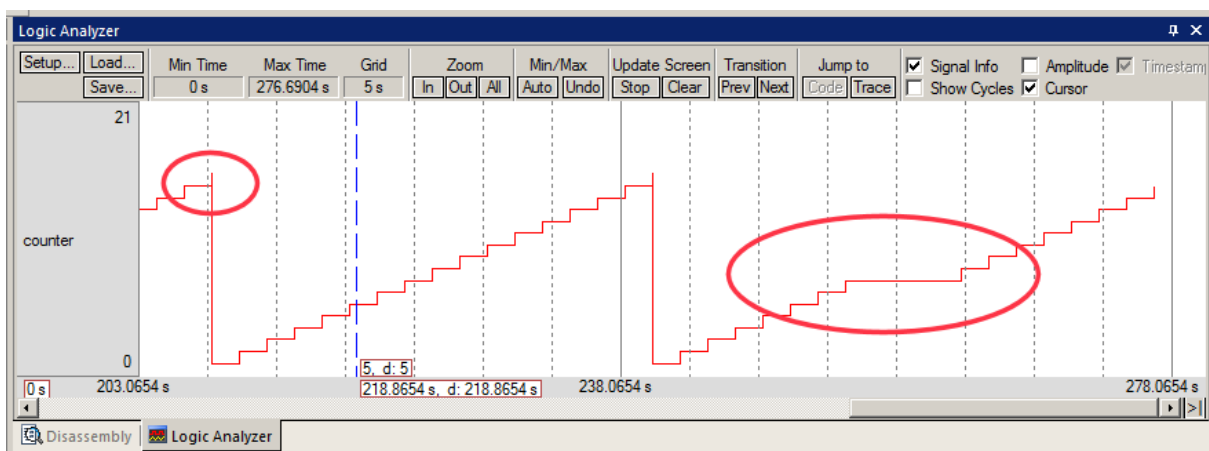
- 1) Click on Zoom Out until Grid is about 5 seconds.
- 2) The variable `counter` will increment to 0x10 (decimal 16) and then set to 0.

**TIP:** If you do not see a waveform, exit and re-enter Debug mode to refresh the LA. You might also have to repower the Xplained board. Confirm the Core Clock: value is correct.

In this project, this Core Clock: value must be ½ the CPU core clock. The trace circuitry in this processor in this project is clocked at half this rate. You can determine the CPU clock speed by viewing the global variable `SystemCoreClock` in a Watch window. The processor clocks are configured in `system_SAMV71.c`.

**TIP:** You can show up to 4 variables in the Logic Analyzer. These variables must be global, static or raw addresses such as \*((unsigned long \*) 0x20000000).












- 3) Press the SW0 User button and see counter increment stop as shown in the circle below:
- 4) Select Signal Info, Show Cycles, Amplitude and Cursor to see the measuring capabilities of the LA. You can stop the LA by clicking on the Stop icon in the Update Screen box.
- 5) Note counter briefly reaches 0x11 since the test is after the increment. This can be very useful to find unusual bugs.
- 6) When you are ready to continue, start the Update Screen.
- 7) Stop the CPU. 
- 8) Exit Debug mode: 

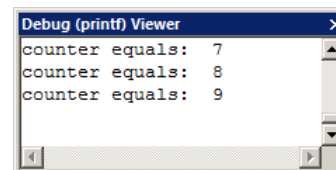
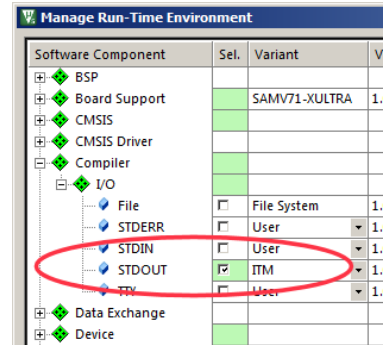


## 15) printf with ITM (Instrumentation Trace Macrocell): ITM uses Serial Wire Viewer:

**This feature is only available with a Keil ULINK2, ULINKpro or a J-Link. EDBG does not currently support SWV.**

It is easy to incorporate printf using ITM and the µVision utility Manage Runtime Environment.

1. Stop the program if it is running  and exit Debug mode. 
2. Open the Manage Run-Time Environment utility.  This window opens:
3. Expand Compiler...I/O.
4. Select STDOUT and ITM as shown here: 
5. All the blocks should be green. If not, click on the Resolve button.
6. Click OK to close this window.
7. The file retarget\_io.c will be added to your project in the project window under the Compiler group.
8. In Blinky.c, near line 20, add `#include <stdio.h>`
9. In Blinky.c, near line 47 just after the `if (counter>....` line, add this line: `printf("counter equals: %d\n", counter);`
10. Select RTX\_Conf\_CM.c, by clicking on its tab. Select the Configuration Wizard tab at its bottom.
11. Expand Thread Configuration. Increase the Main Thread Stack size to 300 bytes. It will change to 296 bytes.
12. Select File/Save All or click .
13. Rebuild the source files .
14. Enter Debug mode . Click on RUN .
15. Select View/Serial Windows and select Debug (printf) Viewer.
16. The values of counter is displayed as seen here: 
17. Stop the program  and exit Debug mode. 



**TIP:** You can easily save ITM information to a file. See [www.keil.com/support/man/docs/uv4/uv4\\_cm\\_itmlog.htm](http://www.keil.com/support/man/docs/uv4/uv4_cm_itmlog.htm)

### Obtaining a character typed into the Debug printf Viewer window:


It is possible for your program to do this with the function `ITM_ReceiveChar` found in `core.CM7.h`.

See [https://www.keil.com/pack/doc/CMSIS/Core/html/group\\_\\_i\\_t\\_m\\_\\_debug\\_\\_gr.html](https://www.keil.com/pack/doc/CMSIS/Core/html/group__i_t_m__debug__gr.html).

A working example can be found in the File System Demo in Keil Middleware. Download this using the Pack Installer utility.

You will need a MDK Professional license to compile and run this middleware component.

### Read-Only Source Files:

Some files in the Project window will have a yellow key on them:  This means they are read-only. This is to help unintentional changes to these source files. This can cause difficult to solve problems. Most of these files will not need any modification in normal use.

If you need to modify a protected file, you can use Windows Explorer to modify its permission.

1. Double click on the file to open it in the Sources window.
2. Right click on its source tab and select Open Containing folder.
3. Explorer will open with the file selected.
4. Right click on the file and select Properties.
5. Unselect Read-only and click OK. You are now able to change the file in the µVision editor.
6. It is a good idea to make the file read-only when you are finished modifications.



## 16) DSP SINE example using ARM CMSIS-DSP Libraries:

ARM CMSIS-DSP libraries are offered for Cortex-M0/M0+, Cortex-M3, Cortex-M4 and Cortex-M7 processors. DSP libraries are provided in MDK in C:\Keil\_v5\ARM\Pack\ARM\CMSIS\\*. CMSIS documentation is located at C:\Keil\_v5\ARM\Pack\ARM\CMSIS\\*. On the web documentation is located here: [www.keil.com/pack/doc/CMSIS/DSP/html/index.html](http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html).

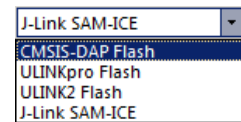
This example creates a sine wave to which noise is added, and then the noise is filtered out leaving the original sine wave.

This example incorporates Keil RTX RTOS. RTX is available free with a BSD type license. Source code is always provided.

**Running the DSP Example:** The DSP example file will have been copied into your computer on page 6.

1. Open the project file sine: C:\00MDK\Boards\Atmel\SAMV71-XULTRA\DSP\sine.uvprojx.

2. Select the target for the debug adapter you are using: Select CMSIS-DAP to use the onboard EDBG adapter as shown here: If you have another adapter listed, use it.



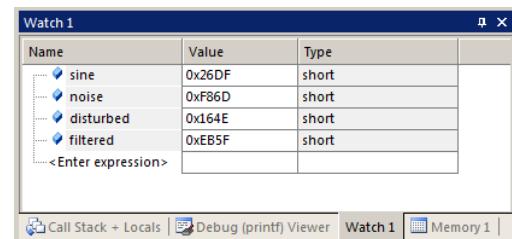
3. Build the files. There will be no errors or warnings.

4. Enter Debug mode by clicking on the Debug icon. The Flash will be programmed.

5. Click on the RUN icon.

6. Open Watch 1 by selecting View/Watch/Watch 1 if necessary.

7. Four global variables will be displayed in Watch 1 as shown here: If these variables are changing the program is working properly.



8. Select View/Serial Windows/Debug (printf) Viewer. Some printf lines are displayed if you are using any ULINK2, ULINKpro or a J-Link. This is implemented with ITM.

**TIP:** Keil documentation uses the term "Thread" instead of "Task" for consistency.

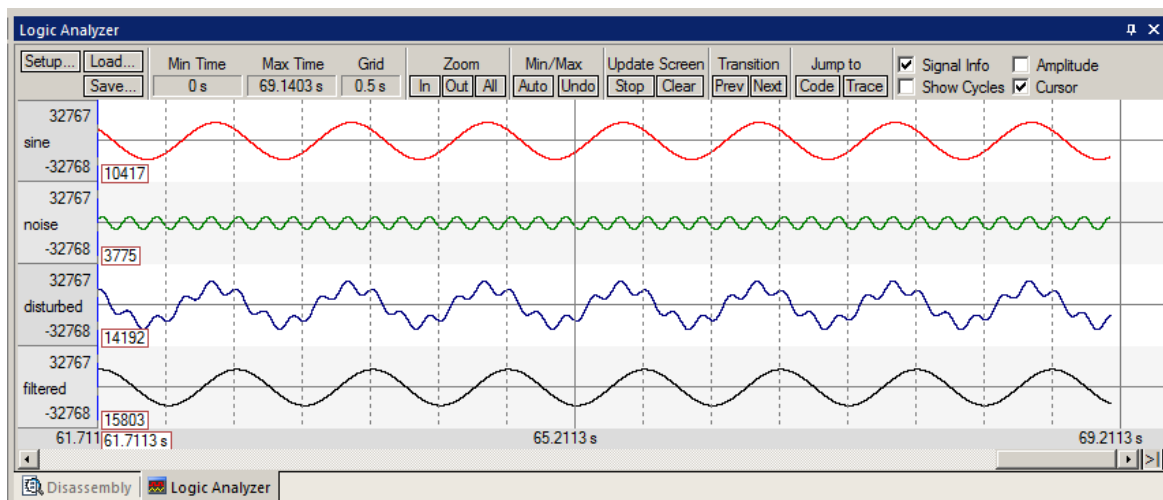
**Serial Wire Viewer (SWV):** (available with a ULINK2, ULINKpro, J-Link or SAM-ICE) EDBG does not yet support SWV.

If you selected either a ULINKpro or ULINK2 as your Target Options (see step 2 above) you will see the Logic Analyzer window below displaying the four global variables displayed in the Watch window.


The Atmel Cortex-M3, Cortex-M4 and Cortex-M7 processors have Serial Wire Viewer (SWV). This is data trace. If you use any Keil ULINK or a J-Link, you can use this Logic Analyzer window plus many other Serial Wire Viewer (SWV) features. Atmel Cortex-M7 processors also have ETM instruction trace. A ULINKpro is needed for this ETM feature.

ETM trace will be shown later in this document.

µC/Probe from Micrium might also be useful to display variables in a graphical format. It is CMSIS-DAP compliant. You do not need to be running a Micrium RTOS to use µC/Probe. See [www.micrium.com/ucprobe/](http://www.micrium.com/ucprobe/)



**RTX System and Thread Viewer:** This feature works with the on-board EDBG, any ULINK or J-Link and SAM-ICE.

1. Click on the RUN icon. 
2. Open Debug/OS Support and select System and Thread Viewer. A window similar to below opens up.
3. This window updates in real-time. Note nearly all the processor time is spent in the idle daemon os\_idle\_demon. The processor spends relatively little time in each thread. You can change this if you need to.
4. Set a breakpoint in one of the threads in DirtyFilter.c by clicking in the left margin on a grey area.
5. Here are the four threads with their approximate starting line numbers:  
 1) sine\_gen (73) 2) noise\_gen (93) 3) disturb\_gen (114) 4) filter\_tsk (135)
6. The program will stop here and the Task window will be updated accordingly. Here, I set a breakpoint in the noise\_gen task: You can see that noise\_gen was running when the breakpoint was activated.
7. os\_idle\_demon is Ready to run when noise\_gen is finished and no other task is Ready.


System and Thread Viewer

Property	Value							
System	Item	Value						
	Tick Timer:	1.000 mSec						
	Round Robin Timeout:							
	Default Thread Stack Size:	296						
	Thread Stack Overflow Check:	Yes						
	Thread Usage:	Available: 6, Used: 6 + os_idle_demon						
Threads	ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
	1	main	Normal	Wait_DLY				cur: 67%, max: 79% [236/296]
	2	sine_gen	Normal	Wait_DLY	1			cur: 27%, max: 32% [96/296]
	3	noise_gen	Normal	Wait_DLY	1			cur: 27%, max: 32% [96/296]
	4	disturb_gen	Normal	Wait_DLY	1			cur: 27%, max: 27% [80/296]
	5	filter_tsk	Normal	Wait_DLY	1			cur: 27%, max: 39% [116/296]
	6	sync_tsk	Normal	Wait_DLY	1			cur: 27%, max: 27% [80/296]
	255	os_idle_demon	None	Running				cur: 0%, max: 21% [0/296]

Trace Data

System and Thread Viewer

**TIP:** os\_idle\_demon has a Priority of 0 which is the lowest priority possible. Every other task has a higher priority.

8. Set another breakpoint in a different task. Click on the RUN icon. 
9. Each time you click on RUN, the program will stop at one of the two tasks and this is indicated by the Running state.
10. Remove the breakpoints by clicking on them or Debug/Breakpoints or Ctrl—B and select Kill All.

**TIP:** Recall this window uses the CoreSight DAP read and write technology to update this window in real-time.



**TIP:** You might have noticed the Event Viewer in Debug/OS Support. This uses Serial Wire Viewer to get its information and this feature is available on Atmel Cortex-M3 , Cortex-M4 and Cortex-M7 processors with any Keil ULINK, J-Link or SAM-ICE (Version 6 or higher) debug adapter. This window is examined on the next page.

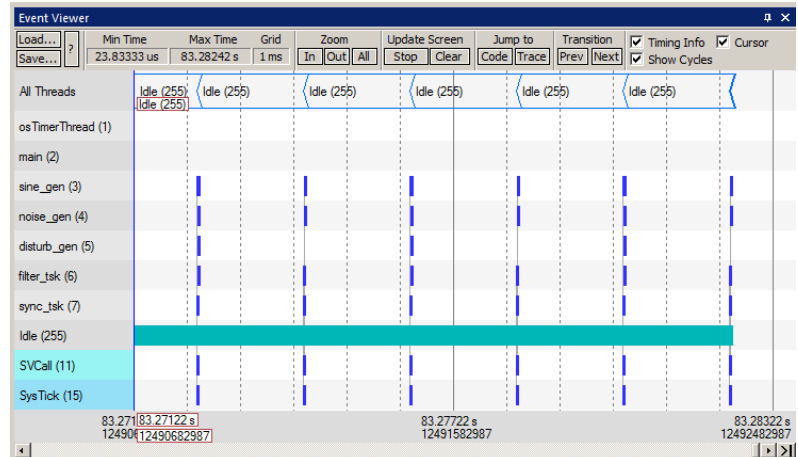
## Event Viewer:

This feature is only available with a Keil ULINK2, ULINKpro, J-Link or a SAM-ICE. EDBG does not yet support SWV.

Serial Wire Viewer (SWV) must be configured for the Event Viewer to work. SWV is pre-configured in this DSP example.

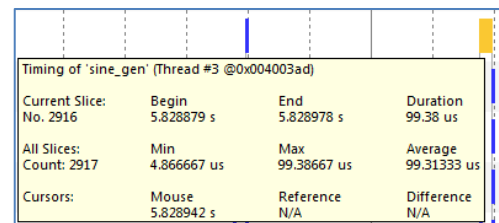
Remember RTX is a free full feature RTOS with a BSD license. All Source Code is provided. See <http://www.keil.com/rtx>

1. Stop the program. 
2. Select Debug/Debug Settings.
3. Click on the Trace tab.
4. Enable ITM Stimulus Port 31. Event Viewer uses this to collect its information.
5. Click OK twice.
6. Click on RUN. 
7. Open Debug/OS Support and select Event Viewer. The window here opens up:



### Key Features of the Event Viewer:

1. Note each thread is plotted. The program spends most of its time in the Idle daemon.
2. If you are using a ULINKpro, the timing of interrupts are also displayed. See SVCALL and Sys Tick above.
3. You can stop/start the update without stopping the program execution. Select Stop on the Update Screen box.
4. Hover your mouse on a block and it turns yellow and displays statistics. The Timing Info box must be enabled:
5. Click on a block on the Y axis it is on and a red line is created. Click on Jump to Code and this area in your source code is highlighted.
6. Select Timing Info. Move your mouse away from the red line and wait a bit for the position to register. A similar box displays statistics about the difference in pointer positions.



The Event Viewer makes it easy to view and adjust the timings of your RTX implementation.

**Serial Wire Viewer Throughput:** If the Event Viewer is empty or displays data that looks incorrect or unstable:

Click on Setup... in the Logic Analyzer. Select Kill All to remove all variables. This is necessary because the SWO pin will likely be overloaded when the Event Viewer is opened up. Inaccuracies might occur.

**TIP:** It is easy to overload the Serial Wire Output pin cause it to drop trace frames. Solutions are to delete some or all of the variables in the Logic Analyzer to free up some bandwidth. You might have to sample your own variables if they change too often. Using a ULINKpro in either Manchester mode or the 4 bit trace port will help.

ULINKpro is much better with SWO bandwidth issues. These have been able to display both the Event and LA windows.

ULINKpro uses the faster Manchester format than the slower UART mode that ST-Link, ULINK2 and J-Link uses.

ULINKpro can also use the 4 bit Trace Port for faster operation for SWV. The Trace Port is mandatory for ETM trace.

If you are using Serial Wire Viewer extensively, the modest investment in a ULINKpro is easily justified.


When there is a SWO or Trace Port overload and frames are lost, µVision recovers gracefully and continues displaying data.

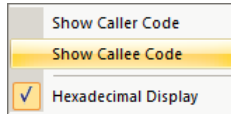
**TIP:** The timestamps values in the LA are derived from the Core Clock: setting in the Trace Configuration window.

**TIP:** ITM Port 31 enables sending the Event Viewer frames out the SWO port. Disabling this can save bandwidth on the SWO port if you are not using the Event Viewer.

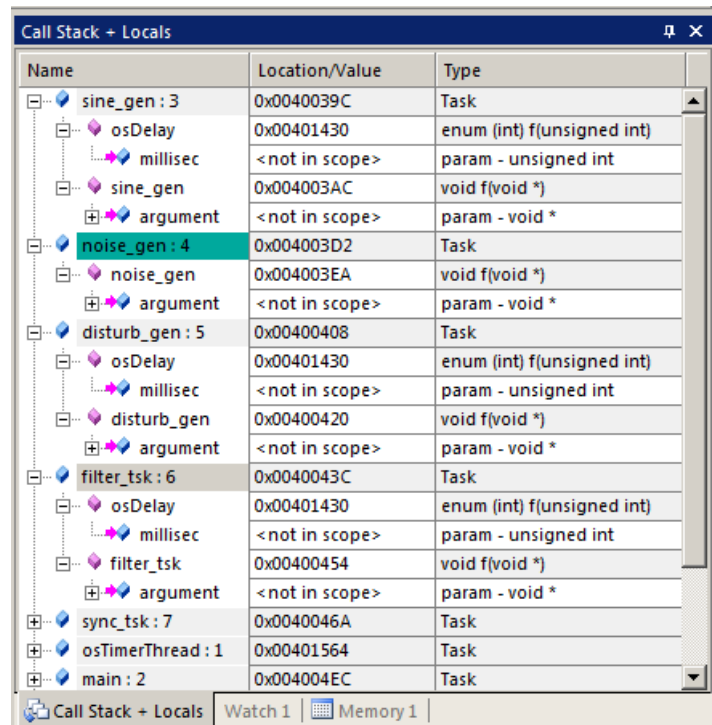
RTX documentation is available here: [www.keil.com/pack/doc/CMSIS/RTX/html/index.html](http://www.keil.com/pack/doc/CMSIS/RTX/html/index.html)

### Call Stack and Locals:

1. Click on the Call Stack + Locals tab. This window opens up: 
2. Each time you stop the program, the information is updated depending on which thread is running.
3. Right click on an element and select Callee or Caller Code to go there:



**TIP:** Recall the Call Stack and Locals window updates only when the program is stopped by one of the two breakpoints that were set on the previous page.



### This is the end of the DSP example.

Using the Manage Run-Time Manager to select various CMSIS components, the correct libraries and sources will be added to your project according to the processor and/or board you selected.

DSP libraries (with source code), examples and documentation are located here: (select your correct CMSIS version number):

C:\Keil\_v5\ARM\Pack\ARM\CMSIS\4.5.0\CMSIS\DSP\_Lib or [www.keil.com/pack/doc/CMSIS/DSP/html/index.html](http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html).

RTX documentation is located here: [www.keil.com/pack/doc/CMSIS/RTX/html/index.html](http://www.keil.com/pack/doc/CMSIS/RTX/html/index.html)



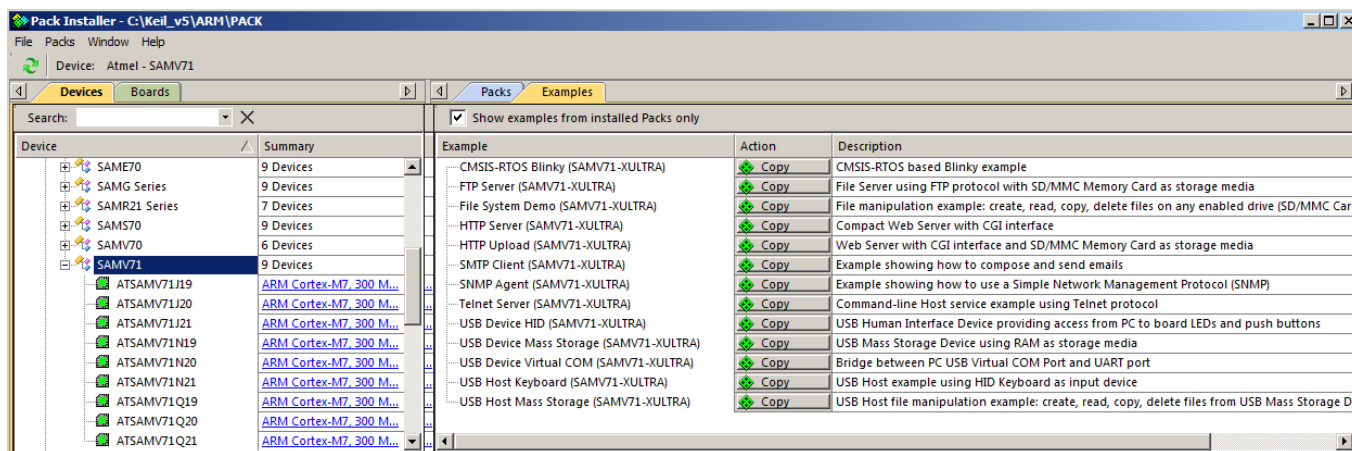
## 17) Keil Middleware: Network: TCP/IP, Flash File, USB, Graphics:

MDK Professional provides commercial grade middleware with extensive capabilities designed for demanding applications. See [www.keil.com/mdk5/middleware/](http://www.keil.com/mdk5/middleware/) for more information.

Working examples are provided for various Atmel boards. You can access and copy these examples using Pack Installer .

Here is the listing of examples for SAMV71:

This window is displayed by selecting SAMV71 in the Devices tab in the Pack Installer:



### License:

An MDK Pro license is needed for Keil Middleware. A 7 day one-time licenses is available in µVision under File/License Management. If you qualify, this button is displayed:

Evaluate MDK Professional

To obtain a temporary MDK Professional license for evaluation purposes, contact Keil sales as listed on the last page.

**Instructions:** Each example contains the file abstract.txt which provides instructions. These projects compile and run "out-of-the-box". If you have any questions regarding Keil Middleware operation during your evaluation phase, please contact Keil Technical Support as listed on the last page of this document.

### Configuring the examples:

The Middleware examples make extensive use of the Configuration Wizard. This permits easy modifications to various settings with mouse clicks instead of digging through source code.

### Selecting the Middleware:

Keil Middleware is selected using the Manage Run-Time Environment utility. This selects the various components you desire and place them into your project. Open the Manage Run-Time Environment utility



and this window is displayed:

### Help and Documentation:

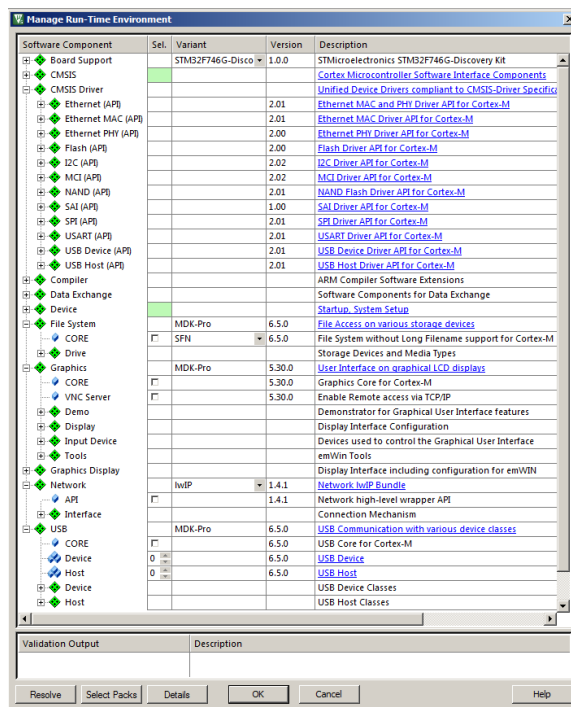
This is available both online and embedded in MDK:

[www.keil.com/pack/doc/mw/General/html/index.html](http://www.keil.com/pack/doc/mw/General/html/index.html)

and

C:\Keil\_v5\ARM\Pack\Keil\MDK-Middleware

On the next page, we will test the File System example.




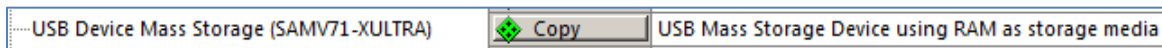
## 18) Keil Middleware Examples:





### A) Running the USB Device Mass Storage example:

**Hardware Preparation:** Install a suitable SD card in the Xplained board:

You can find additional instructions here: [http://www.keil.com/pack/doc/MW/USB/html/dev\\_msc\\_tutorial.html](http://www.keil.com/pack/doc/MW/USB/html/dev_msc_tutorial.html)

1. Open the Pack Installer. 
2. In the Devices tab, select SAMV71. This filters the right side of the Pack Installer.
3. Select the Examples tab and copy USB Device Mass Storage into C:\00MDK\



4. In  $\mu$ Vision, open the MassStorage project:  
C:\00MDK\Boards\Atmel\SAMV71-XULTRA\Middleware\USB\Device\MassStorage\MassStorage.uvprojx
5. If you are using the on-board EDBG CMSIS-DAP adapter: select SAMV7 Flash-DAP: 
6. Connect your PC USB cable to the DEBUG USB connector.
7. If you are using another adapter, you will have to make suitable adjustments to these instructions.
8. Build the files.  There will be no errors or warnings.
9. Enter Debug mode:  Click on the RUN icon. 
10. Connect another USB cable to the connector TARGET USB.
11. You will hear the USB enumerate double tone.
12. Open Microsoft Explorer and you will find an entry for your SD card listed in the usual manner.
13. If you do not see this or the Device is not recognized, cycle the USB connections.
14. If the green LED POWER blinks, supply 5 volts power to the barrel VIN connector or try the Debug USB connector.  
It is possible when using the Xplained board USB port it will exceed the power capabilities of your PC.

**TIP:** The program is now programmed into the SAMV7 Flash and will run stand-alone without  $\mu$ Vision connected.

### B) Running the File System File example:

This example creates a drive using an SD card in the SAMV71 Xplained board using the Keil You can create, read, copy and delete files on any enabled drive (SD/MMC Card, NOR/NAND Flash) and format each drive.

A terminal program is created using the Serial Wire Viewer ITM channel. This means this example is best used with a ULINK2, ULINK $\mu$ , SAM-ICE or a Segger J-Link.

**TIP:** If you are intreted in using a two-way terminal program with uVision and the Xplained board, examine this example to see how to accomplish this.

As mentioned in the Abstract.txt file, detailed instructions can be found here:

[www.keil.com/pack/doc/MW/FileSystem/html/fs\\_examples.html#fs\\_standalone\\_example](http://www.keil.com/pack/doc/MW/FileSystem/html/fs_examples.html#fs_standalone_example)

## 19) Creating your own MDK 5 project from scratch:

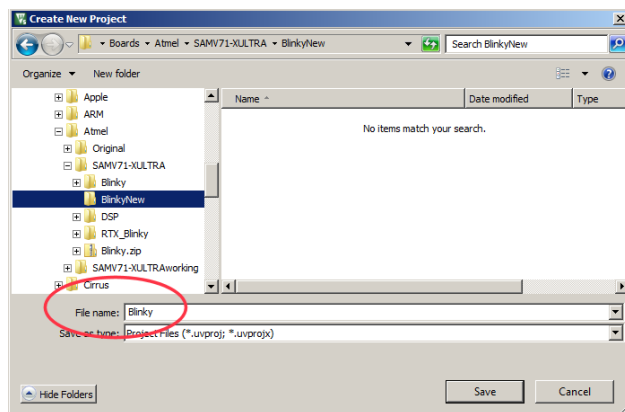
All examples provided by Keil are pre-configured. All you have to do is compile them. You can use them as a starting point for your own projects. However, we will start a project from the beginning to illustrate how easy this process is. Once you have the new project configured; you can build, load and run your example. It will have an empty main() function so it does not do much. However, the processor startup sequences are present and you can easily add your own source code and/or files. You can use this process to create any new project, including one using an RTOS as shown on page 26.

### Install the Software Pack for your processor:

1. Start  $\mu$ Vision and leave it in Edit mode. Do not be in Debug mode.
2. **Pack Installer:** The Software Pack for your processor must be installed. This has already been done on page 5.
3. You do not need to copy any examples over.

### Create a new Directory and a New Project:

1. In the main  $\mu$ Vision menu, click on Project/New  $\mu$ Vision Project...
2. In the window that opens, go to the folder C:\00MDK\Boards\Atmel\SAMV71-XULTRA
3. Right click in this window and select New and create a new folder. I called it BlinkyNEW.
4. Double click on BlinkyNew to open it or highlight it and select Open.
5. In the File name: box, enter Blinky. Click on Save.
6. This creates the project Blinky.uvproj.
7. As soon as you click on Save, the next window opens:




### Select the Device you are using:

1. Expand Atmel SAMV71 and then select ATSAMV71Q21 for your processor as shown here:

**TIP:** Chip icons in colour are from MDK 5 Software Packs. Grey icons are from MDK 4.7x.

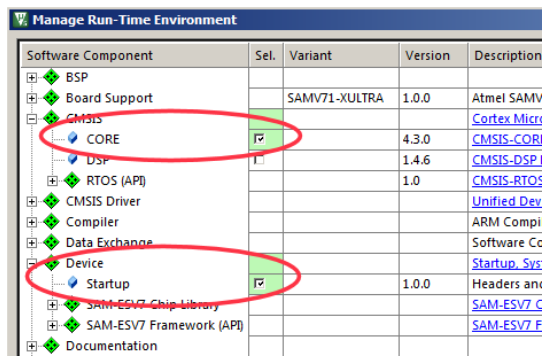
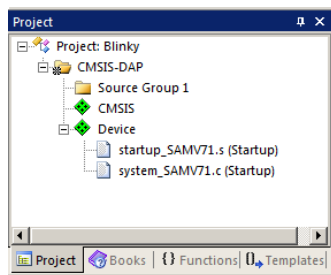
2. Click OK and the Manage Run Time window shown below bottom right opens.

### Select the CMSIS components you want:

1. Expand CMSIS and Device. Select CORE and Startup as shown below right. They will be highlighted in Green indicating there are no other files needed. Click OK to close.
2. Click on File/Save All or select the Save All icon: 
3. The project Blinky.uvproj will now be changed to Blinky.uvprojx.
4. You now have a new project list as shown on the bottom left below: The appropriate CMSIS files you selected have been automatically entered and configured into your project for your selected processor.
5. Note the Target Selector says Target 1. Highlight Target 1 in the Project window.
6. Click once on it and change its name to CMSIS-DAP and press Enter. The Target selector name will also change.

### What has happened to this point:


You have created a blank  $\mu$ Vision project using MDK 5 Software Packs. All you need to do now is add your own source files.

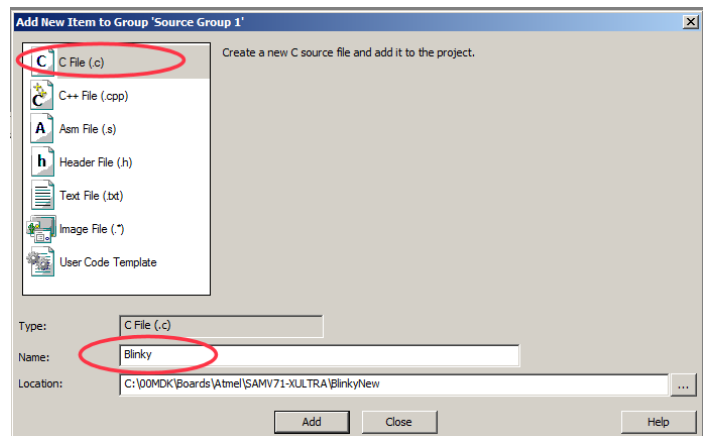


### Create a blank C Source File:

1. Right click on Source Group 1 in the Project window and select

Add New Item to Group 'Source Files'...


2. This window opens up:
3. Highlight the upper left icon: C file (.c):
4. In the Name: field, enter Blinky.
5. Click on Add to close this window.
6. Click on File/Save All or 
7. Expand Source Group 1 in the Project window and Blinky.c will now display.
8. It will also open in the Source window.



### Add Some Code to Blinky.c:

9. In the blank Blinky.c, add the C code below:

10. Click on File/Save All or 

11. Build the files.  There will be no errors or warnings if all was entered correctly.

```
#include "sam.h" // Device header
#include "RTE_Components.h" // Component selection


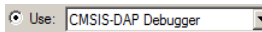
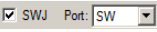

unsigned int counter = 0;
/*-----
  MAIN function
  -----*/
int main (void) {

    while(1) {
        counter++;
        if (counter > 0x0F) counter = 0;
    }
    //make sure you add a CR Carriage Return or Enter after the last parentheses.
```

**TIP:** You can also add existing source files:


Add Existing Files to Group 'Source Files'...

### Configure the Target CMSIS-DAP: *Please complete these instructions carefully to prevent unusual problems...*

1. Select the Target Options icon . Select the **Target** tab. Note the Flash and RAM addresses are already entered.
2. Click on the **Debug** tab. Select CMSIS-DAP Debugger in the Use: box:  Settings
3. Select Settings: icon beside Use: CMSIS-DAP.
4. Set SWJ and SW as shown here:  If your board is connected to your PC and Debug USB connector, you should now see a valid IDCODE and Device Name in the SW Device box.
5. Click on OK **once** to go back to the Target Configuration window.
6. Click on the **Utilities** tab. Select Settings and confirm the correct Flash algorithm is present: Shown are the correct ones for the SAMV71 Xplained board: 
7. Click on OK twice to return to the main menu.

Programming Algorithm			
Description	Device Size	Device Type	Address Range
ATSAMV7x 2048kB Flash	2M	On-chip Flash	00400000H - 005FFFFFFH

8. Click on File/Save All or 

9. Build the files.  There will be no errors or warnings if all was entered correctly. If there are, please fix them !

**The Next Step ? First we will do a summary of what we have done so far and then....**





**Let us run your program and see what happens ! Please turn the page....**

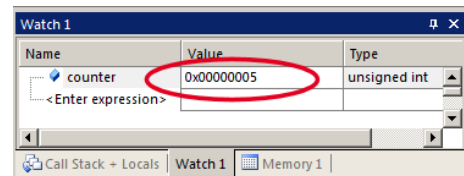
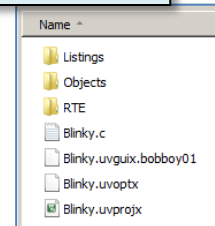


### What we have so far ?

1. A MDK 5 project has been created in C:\00MDK\Boards\Atmel\SAMV71-XULTRA \BlinkyNEW
2. The folders have been created as shown here below:
3. RTE contains the CMSIS-Core startup and system files.
4. The Software Pack has automatically pre-configured many items in your new project.

### Running Your Program:

1. Enter Debug mode by clicking on the Debug icon  The Flash will be programmed.
2. Click on the RUN icon  **Note:** you stop the program with the STOP icon 
3. Right click on counter in Blinky.c and select Add counter to ... and select Watch 1.
4. counter should be updating as shown here: 
5. You can also set a breakpoint in Blinky.c and the program should stop at this point if it is running properly. If you do this, remove the breakpoint.
6. You should now be able to add your own source code to create a meaningful project.



**TIP:** Watch 1 is updated periodically, not when a variable value changes. Since Blinky is running very fast without any time delays inserted, the values in Watch 1 will appear to jump and skip sequential values you know must exist.

### Configuring the CPU Clock and Enabling the Caches:

The CMSIS-CORE file system\_SAMV71.c contains CPU clock configuration code with a global variable CoreSystemClock to indicate the CPU frequency. We will run the two functions in this file to configure the clock.

1. In Blinky.c, near line 10, add these lines:  
Add these before the while loop in main().

```
10      SystemCoreClockUpdate();  
11      SCB_EnableDCache();  
12      SCB_EnableICache();
```


2. Click on File/Save All or 

12. Build the files.  There will be no errors or warnings.

13. Enter Debug mode by clicking on the Debug icon  The Flash will be programmed. **Do not click RUN yet !**

14. In Watch 1, double click on Enter Expression and enter SystemCoreClock. Press Enter.

15. Right click on SystemCoreClock and unselect Hexadecimal Display. The value of 4000000 will be displayed. This is the initial CPU clock speed of 4 MHz.

16. Click on the RUN icon  The clock increases to 300 MHz with the execution of the clock files.

17. The Data and Instruction caches are also enabled by calling the two CMSIS functions listed in Step 1.

18. Stop the CPU.  and exit Debug mode. 






### What else can we do ?

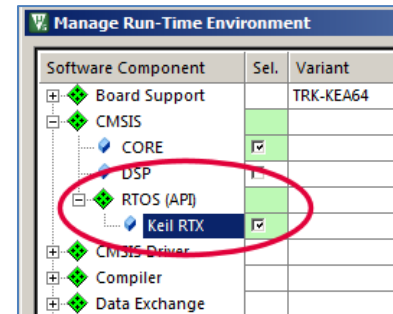
5. You can add your source files using the Add New Item window. See the top of the previous page.
6. You can add existing source files by right clicking on a Group name and selecting Add Existing Item.
7. If you use RTX or Keil middleware, source and template files are provided in the Add New window.
8. Atmel | START can be used to create entire  $\mu$ Vision projects. <http://start.atmel.com>

## 20) Creating your own RTX MDK 5 project from scratch:


The MDK Software Packs contains RTX software components. RTX is CMSIS-RTOS compliant.

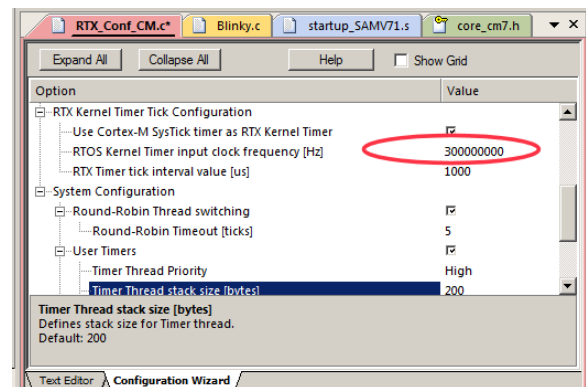
Creating and configuring RTX is easy in MDK 5. These steps use the preceding BlinkyNEW example you constructed.

1. Using the same example from the preceding pages, Stop the program  and exit Debug mode. 
2. Open the Manage Run-Time Environment window: 
3. Expand all the elements as shown here: 
4. Select Keil RTX as shown and click OK.
5. Appropriate RTX files will be added to your project. See the Project window.
6. In Blinky.c, right click near line 3 and select Insert '# include file'. Select "cmsis\_os.h". This will be added to Blinky.c.
7. Click on File/Save All or 






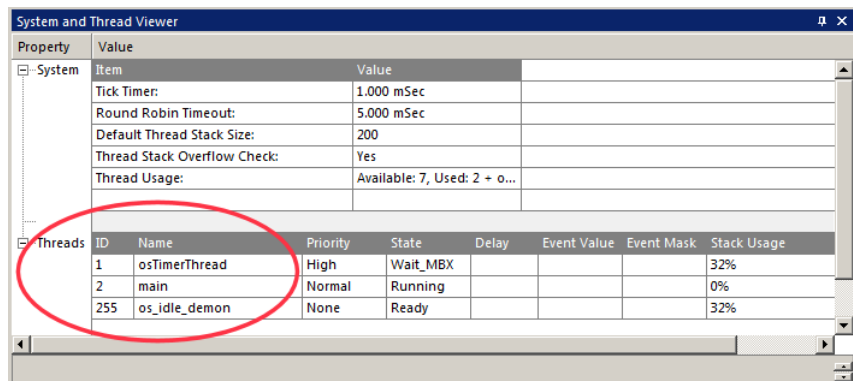
### Configure RTX:

1. In the Project window, expand the CMSIS group.
2. Double click on RTX\_Conf\_CM.c to open it.
3. Select the Configuration Wizard tab: Select Expand All.
4. The window is displayed here: 
5. Set RTOS Kernel Timer clock value: to 300 MHz as shown:
6. Use the defaults for the other settings.



### Build and Run Your RTX Program:

1. Build the files.  There will be no errors or warnings.
2. Enter Debug mode:  Click on the RUN icon. 
3. Select Debug/OS Support/System and Thread Viewer. The window below opens up:
4. You can see three threads: the main thread is the only one running. As you add more threads to create a real RTX program, these will automatically be added to this window.



### What we have so far ?

1. You must add the RTX framework into your code and create your threads to make this into a real RTX project configured to your needs. See the next page.
2. **Getting Started MDK 5:** Obtain this useful book here: [www.keil.com/mdk5/](http://www.keil.com/mdk5/). It has very useful information on implementing RTX.
3. **RTX docs are located here:** [www.keil.com/pack/doc/CMSIS/RTOS/html/index.html](http://www.keil.com/pack/doc/CMSIS/RTOS/html/index.html)

**TIP:** The Configuration Wizard is a scripting language as shown in the Text Editor as comments starting such as a </h> or <i>. See [www.keil.com/support/docs/2735.htm](http://www.keil.com/support/docs/2735.htm) for instructions to add this to your own source code.

## 21) Adding a Thread to your RTX\_Blinky:

We will create and activate a thread. We will add another variable counter2 to give it something to do.

1. In Blinky.c, add this line near line 6: unsigned int counter2=0;

```
6 unsigned int counter2=0;
```

### Create the Thread job1:

2. Add this code to be the thread job1:  
Add this before the main() function.  
osDelay(500) delays the program by 500 clock ticks to slow it down so we can see the values of both counter and counter2 increment by 1.

```
8 void job1 (void const *argument) {  
9     for (;;) {  
10         counter2++;  
11         if (counter2 > 0xf) counter2=0;  
12         osDelay(500);  
13     }  
14 }
```

### Add osDelay to main():





3. Add this line just after the if statement near line 27: 27 osDelay(500);

### Define and Create the Thread:

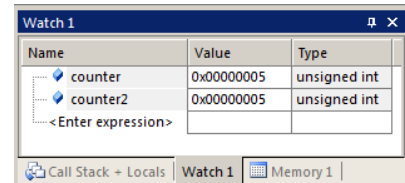
4. Add this line near line 15 just before main():
5. Create the thread job1 near line 23 just before the while(1) loop:

```
16 osThreadDef(job1, osPriorityNormal, 1, 0);
```

```
26 osThreadCreate(osThread(job1), NULL);
```

6. Click on File/Save All or 
7. Build the files.  There will be no errors or warnings. If there are, please fix them before continuing.
8. Enter Debug mode:  Click on the RUN icon. 
9. Right click on counter2 in Blinky.c and select Add counter2 to ... and select Watch 1.

10. Both counter and counter2 will increment but slower than before:  
The two osDelay(500) function calls each slow the program down by 500 msec. This makes it easier to watch these two global variables increment.  
osDelay() is a function provided by RTX.



Name	Value	Type
counter	0x00000005	unsigned int
counter2	0x00000005	unsigned int
<Enter expression>		

11. Open the System and Thread Viewer by selecting Debug/OS Support.
12. Note that job1 has now been added as a thread as shown below:
13. Note os\_idle\_demon is always labelled as Running. This is because the program spends most of its time there.
14. Set a breakpoint in job1 and the program will stop there and job1 is displayed as "Running" in the Viewer.
15. Set another breakpoint in the while(1) loop in main() and each time you click RUN, the program will change threads.
16. There are many attributes of RTX you can add. RTX help files are located here depending on your CMSIS version:  
C:/Keil\_v5/ARM/Pack/ARM/CMSIS/4.3.0/CMSIS/Documentation/RTX/html/index.html.
17. This concludes the DSP lab. Next is ETM Instruction Trace. You will need a ULINKpro for this section.

System and Thread Viewer

Property	Value							
System	Item	Value						
	Tick Timer:	1.000 mSec						
	Round Robin Timeout:	5.000 mSec						
	Default Thread Stack Size:	200						
	Thread Stack Overflow Check:	Yes						
	Thread Usage:	Available: 7, Used: 4 + o...						
Threads	ID	Name	Priority	State	Delay	Event Value	Event Mask	Stack Usage
	1	osTimerThread	High	Wait_MBX				36%
	2	main	Normal	Wait_DLY	370			36%
	3	job1	Normal	Wait_DLY	370			36%
	255	os_idle_demon	None	Running				

## 22) ETM Trace with ULINKpro:

### Introduction:

The examples shown previously with the ULINK2 will also work with the ULINKpro. There are two major differences:

- 1) The window containing the trace frames is now called Trace Data. More complete filtering is available.
- 2) With the ULINK2, SWV (Serial Wire Viewer) data is sent out the 1 bit SWO pin using UART encoding. The ULINKpro can send SWV data *either* out this same SWO pin using Manchester encoding or through the 4 bit Trace Port. This allows ULINKpro to support those Cortex-M processors that have SWV but no Trace Port. The Trace Port is found on the 20 pin Cortex connector and is configured in the Trace configuration window. ETM frames are always sent out the Trace Port and if this is the case, SWV data is also be sent out this port.

### ULINKpro offers:

- 1) Faster Flash programming than the ULINK2 or EDBG adapters.
- 2) All Serial Wire Viewer features that ULINK2 provides but at a much faster data throughput.
- 3) Provides ETM Instruction Trace which provides a record of all executed instructions.
- 4) The Trace Data window has Trace start and stop, filtering and ability to save records to a file. (*in development*)
- 5) **Code Coverage:** Were all the assembly instructions executed ? Useful for program certification.
- 6) **Performance Analysis:** Displays where the processor spent its time in graphical and numerical formats.
- 7) **Execution Profiling:** How long instructions, ranges of instructions, functions or C source code took in both time and CPU cycles as well as number of times these were executed.

## 23) Configuring ULINKpro ETM Trace:



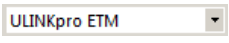

### Configuring the Connection:

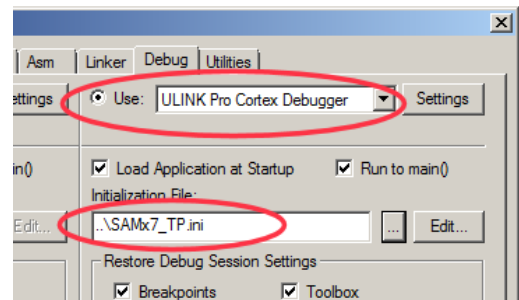
The ULINKpro was configured for SWV operation using the SWO pin and Manchester encoding on page 14. We will activate ETM trace in the next few pages. We will output the trace frames, including SWV, out the 4 bit Trace Port.

### .ini File:

A script must be executed upon entering Debug mode to configure the ETM registers and GPIO ports. This ASCII script is SAMx7\_TP.ini and a copy is found in C:\00MDK\Boards\Atmel\SAMV71-XULTRA\ as installed with the examples with this tutorial. This is an ASCII file. This file is specific to SAMV7x processors as their GPIO ports must be configured.

### Entering the Initialization File:


- 1) Select Project/Open Project. Open the file C:\00MDK\Boards\Atmel\SAMV71-XULTRA\Blinky\Blinky.uvprojx.
- 2) **Create a new Target Options configuration:** Select Project/Manage/Project Items: 
- 3) Click on the Insert icon:  Enter **ULINKpro ETM** (or something of your choice). Enter and then Close
- 4) Select "Ulinkpro ETM": 
- 5) Click on the Target Options icon 
- 6) Click on the Debug tab.
- 7) Select ULINK Pro Cortex Debugger as shown here:
- 8) Enter the SAMx7\_TP.ini in the Initialization file: box and shown here: Use the browse icon to select it from C:\00MDK\Boards\Atmel\SAMV71-XULTRA\
- 9) If you click on the Edit icon, TracePort.ini will be opened with the other source files. You can then view and edit it.
- 10) Leave this Debug tab open for the next page.

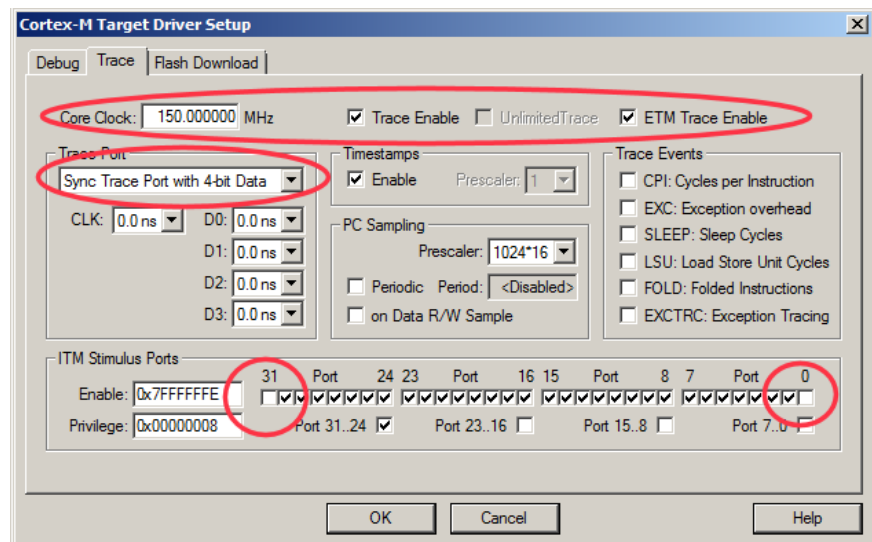


**TIP:** This ini file will be executed every time you enter Debug mode. In the case of this file, a µVision RESET will run it again because of the function OnResetExec. See [www.keil.com/support/man/docs/uv4/uv4\\_db\\_trace\\_init.htm](http://www.keil.com/support/man/docs/uv4/uv4_db_trace_init.htm) for more information.



The next page describes how to configure ETM.

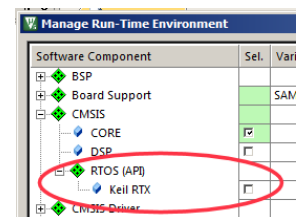
## Configuring ETM Trace:

- 1) These instructions assume you are continuing from the previous page.
- 2) In the Target Options window left open at the Debug tab from the previous page, click on Settings: beside the name of your adapter (ULINK Pro Cortex Debugger) on the right side of the window.
- 3) Click on the Trace tab. The window below is displayed.
- 4) Select Trace Enable.
- 5) In Core Clock: enter a speed of 150 MHz.  $\mu$ Vision uses this to calculate displayed timestamps. This is actually the Trace (TPIU) input clock rather than the actual CPU speed of 300 MHz in this case.
- 6) In Trace Port select **Sync Trace Port with 4 bit Data** as shown below.
- 7) Select ETM Trace Enable.
- 8) Unselect EXCTRC, ITM 0 and 31. Leave everything else at default as shown below.
- 9) Click on OK twice to return to the main  $\mu$ Vision menu. ETM is now configured through the 4 bit Trace Port.
- 10) Select File/Save All. 



## Configure Software Attributes:

1. Open Manage Run-Time Environment: 
2. Unselect Keil RTX as shown here: We do this to simplify our example.  $\mu$ Vision can trace all instructions through any RTOS such as RTX no matter how complicated.
3. Click on OK to close Manage Run-Time Environment window.
4. In Blinky.c do these to further simplify our trace window example:
  - a. Comment out this line: `#include "cmsis_os.h"` (near line 16).
  - b. Comment out both Cache enable function calls just inside the main() function near lines 36 and 37.
  - c. There are two lines `osDelay(500);` Comment both of these function calls out. These are part of RTX which has now been disabled.
5. Select File/Save All or .



The Atmel SAM V7 has JTAG support only for boundary scan use and not for debugging. SWD (Serial Wire Debug) is provided for debugging.



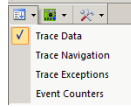

Be aware the five Trace Port pins are normally multiplexed with GPIO pins or other peripherals. You should make appropriate allowances for the use of these shared ports during debugging with ETM trace.

**TIP:** It is good engineering practice *during system design* to not use those pins shared with ETM for important purposes that will preclude normal operation with ETM enabled.



## 24) Blinky Example: ETM Frames starting at RESET and beyond:

The Blinky project has now been modified on the previous page to provide ETM Trace and the features it provides.

- Power up the Xplained board and the ULINKpro *in this order*:
  - Both the Xplained board and ULINKpro are not powered at this point.
  - Connect the ULINKpro 20 pin cable to the Coresight ETM & SWD connector on the Xplained board.
  - Power the Xplained board using USB cable at TARGET USB connector or VIN connector and then.....
  - Finally, power the ULINKpro.
  - If these steps are not followed, you can get trace errors and overruns. We are investigating this.
- Compile the Blinky source files by clicking on the Rebuild icon. . You can also use the Build icon beside it.
- Enter Debug mode by clicking on the Debug icon.  Select OK if the Evaluation Mode box appears.
- DO NOT CLICK ON RUN YET !!! If you did, simply exit and re-enter Debug mode.
- Open the Trace Data window by clicking on the small arrow beside the Trace icon as shown here: 
- Examine the Trace Data window as shown below: This is a complete record of all the program flow since RESET until µVision halted the program at the start of main() since Run To main is selected.
- In this case, the last instruction to be executed is. (BL.W). main In the Register window the PC will display the value of the next instruction to be executed (0x0040 05FC in my case). Click on Single Step once. 

Trace Data				
Display: All				
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x00400256	MOV r6,#0x00		__user_setup_stackheap
	X: 0x0040025A	MOV r7,#0x00		__user_setup_stackheap
	X: 0x0040025E	MOV r8,#0x00		__user_setup_stackheap
	X: 0x00400262	MOV r11,#0x00		__user_setup_stackheap
	X: 0x00400266	BIC r1,r1,#0x07		__user_setup_stackheap
	X: 0x0040026A	MOV r12,r5		__user_setup_stackheap
	X: 0x0040026C	STM r12,[r6-r8,r11]		__user_setup_stackheap
	X: 0x00400270	STM r12,[r6-r8,r11]		__user_setup_stackheap
	X: 0x00400274	STM r12,[r6-r8,r11]		__user_setup_stackheap
	X: 0x00400278	STM r12,[r6-r8,r11]		__user_setup_stackheap
	X: 0x0040027C	MOV sp,r1		__user_setup_stackheap
0.000 226 807 s	X: 0x0040027E	BX lr		__user_setup_stackheap
	X: 0x004001C4	MOV r1,r2		???
0.000 227 173 s	X: 0x004001C6	BLW __rt_lib_init (0x004001B4)		???
	X: 0x004001B4	PUSH {r0-r4,lr}		???
0.000 227 453 s	X: 0x004001B6	BLW _fp_init (0x0040065C)		???
	X: 0x0040065C	MOV r0,#0x3000000		__fp_init
	X: 0x00400660	VMSR FPSCR,r0		__fp_init
0.000 227 960 s	X: 0x00400664	BX lr		__fp_init
0.000 228 347 s	X: 0x004001BA	POP {r0-r4,pc}		???
0.000 228 640 s	X: 0x004001CA	BLW main (0x004005FC)		???

- The next instruction BL.w will display at 0x0040 05FC which is the first instruction inside main().

0.000 228 640 s	X: 0x004001CA	BLW main (0x004005FC)		???
0.000 229 113 s	X: 0x004005FC	BLW LED_GetCount (0x004002E0)	int32_t max_num = LED_GetCount() - 1;	main

- Scroll to the top of the Trace Data window to the first frame. This is the first instruction executed after the initial RESET sequence. In this case it is a LDR as shown below:

- If you use the Memory window to look at location 0x4, you will find the address of the first instruction there and this will match with that displayed in frame # 1. In my case it is 0x0040 01E1 - 1 = 0x0040 01E0 (+1 says it is a Thumb instruction).

These first instructions after RESET are shown below: Note any source information available is displayed:

Trace Data				
Display: All				
Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x004001E0	LDR r0,[pc,#36] ; @0x00400208	LDR R0,=SystemInit	Reset_Handler
0.000 000 000 s	X: 0x004001E2	BLX r0	BLX R0	Reset_Handler
	X: 0x00400518	LDR r0,[pc,#188] ; @0x004005D8	SCB->CPACR  = (3UL << 10*2)	SystemInit
	X: 0x0040051A	LDR r0,[r0,#0x00]		SystemInit

**TIP:** If you double-click on any trace line, this will be highlighted in both the Disassembly and source windows.

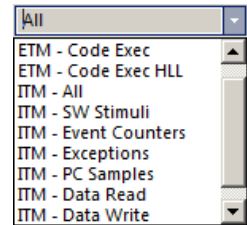
**ETM trace provides a powerful tool for finding nasty bugs not easily found any other way. See page 36.**

## Finding the Trace Frames you are looking for:

Capturing all the instructions executed is possible with ULINK<sub>pro</sub> but this might not be practical. It is not easy sorting through millions and billions of trace frames or records looking for the ones you want. You can use Find, Trace Triggering, Post Filtering or save everything to a file and search with a different application program such as a spreadsheet.

### Trace Filters:

In the Trace Data window you can select various types of frames to be displayed. Open the Display: box and you can see the various options available as shown here: These filters are post collection. Future enhancements to  $\mu$ Vision will allow more precise filters to be selected.




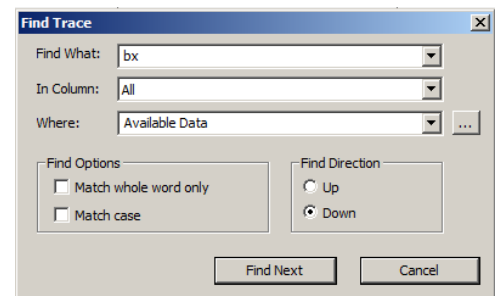
### Find a Trace Record:

In the Find a Trace Record box enter bx as shown here:



Note you can select properties where you want to search in the “in” box. All is shown in the screen above

Select the Find a Trace Record icon  and the Find Trace window screen opens as shown here: Click on Find Next and each time it will step through the Trace records highlighting each occurrence of the instruction bx.



### Trace Triggering: *coming soon for Atmel Cortex-M7*

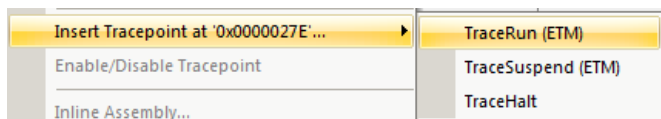
$\mu$ Vision has three trace triggers currently implemented:

**TraceRun:** Starts ETM trace collection when encountered.

**TraceSuspend:** Stops ETM trace collection when encountered. TraceRun has to have been encountered for this to have an effect.

These two commands have no effect on SWV or ITM. TraceRUN starts the ETM trace and TraceSuspend stops it.

**TraceHalt:** Stops ETM trace, SWV and ITM. Can be resumed only with a STOP/RUN sequence. TraceStart will not restart this.





### How it works:

When you set a TraceRun point in assembly language point, ULINK<sub>pro</sub> will start collecting trace records. When you set a TraceSuspend point, trace records collection will stop there. EVERYTHING in between these two times will be collected. This includes all instructions through any branches, exceptions and interrupts.








## How to Set Trace Triggers:

***Coming Soon for Atmel SAM7 Cortex-M7 Processors:***

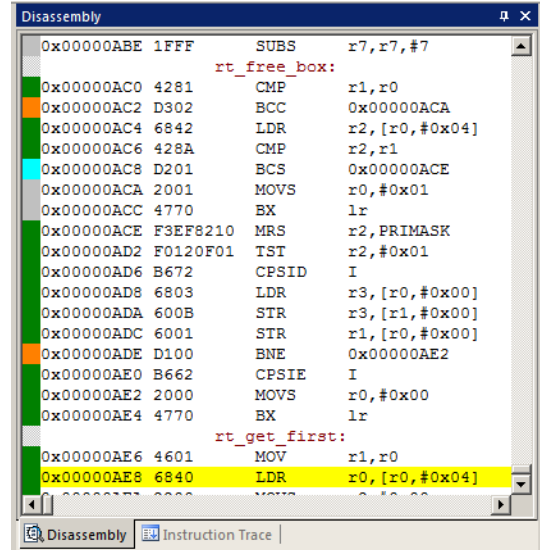
## 25) Code Coverage:

1. Click on the RUN icon.  After a second or so stop the program with the STOP icon. 
2. Examine the Disassembly and Blinky.c windows. Scroll and notice different color blocks in the left margin:
3. This is Code Coverage provided by ETM trace. This indicates if an instruction has been executed or not.

Colour blocks indicate which assembly instructions have been executed.

- |   |   |
|---|---|
|  | 1. Green: this assembly instruction was executed.     |
|  | 2. Gray: this assembly instruction was not executed.  |
|  | 3. Orange: a Branch is always not taken.              |
|  | 4. Cyan: a Branch is always taken.                    |
|  | 5. Light Gray: there is no assembly instruction here. |
|  | 6. RED: A hardware breakpoint is set here.            |
|  | 7. The PC: The next instruction to be executed.       |

In the window on the right you can easily see examples of each type of Code Coverage block and if they were executed or not and if branches were taken (or not).



```

Disassembly
0x00000ABE 1FFF SUBS    r7,r7,#7
                                rt_free_box:
0x00000AC0 4281 CMP     r1,r0
0x00000AC2 D302 BCC     0x00000ACA
0x00000AC4 6842 LDR     r2,[r0,#0x04]
0x00000AC6 428A CMP     r2,r1
0x00000AC8 D201 BCS     0x00000ACE
0x00000ACA 2001 MOVS    r0,#0x01
0x00000ACC 4770 BX      lr
0x00000ACE F3EF8210 MRS     r2,PRIMASK
0x00000AD2 F0120F01 TST     r2,#0x01
0x00000AD6 B672 CPSID   I
0x00000AD8 6803 LDR     r3,[r0,#0x00]
0x00000ADA 600B STR     r3,[r1,#0x00]
0x00000ADC 6001 STR     r1,[r0,#0x00]
0x00000ADE D100 BNE     0x00000AE2
0x00000AE0 B662 CPSIE   I
0x00000AE2 2000 MOVS    r0,#0x00
0x00000AE4 4770 BX      lr
                                rt_get_first:
0x00000AE6 4601 MOV     r1,r0
0x00000AE8 6840 LDR     r0,[r0,#0x04]
  
```

**TIP:** Code Coverage is visible in both the disassembly and source code windows. Click on a line in one and this place will be matched in the other.

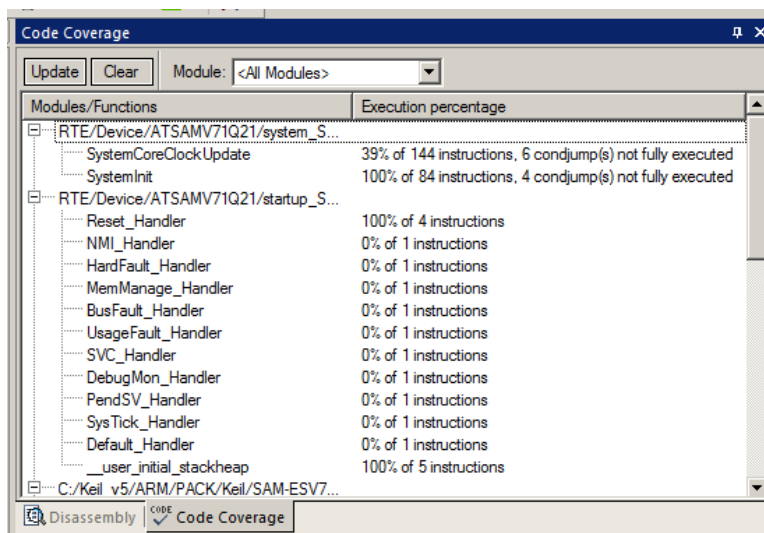
Why was 0x0000\_0ACA never executed ? You should devise tests to execute instructions that have not been executed. What will happen to your program if this untested instruction is unexpectedly executed ?

Code Coverage tells what assembly instructions were executed. It is important to ensure all assembly code produced by the compiler is executed and tested. You do not want a bug or an unplanned circumstance to cause a sequence of untested instructions to be executed. The result could be catastrophic as unexecuted instructions have not been tested. Some agencies such as the US FDA require Code Coverage for certification.

Good programming practice requires that these unexecuted instructions be identified and tested.

Code Coverage is captured by the ETM. Code Coverage is also available in the Keil Simulator.

A Code Coverage window is available as shown below. This window is available in View/Analysis/Code Coverage. The next page describes how you can save Code Coverage information to a file.



Modules/Functions	Execution percentage
RTE/Device/ATSAMV71Q21/system_S...	
SystemCoreClockUpdate	39% of 144 instructions, 6 condjump(s) not fully executed
SystemInit	100% of 84 instructions, 4 condjump(s) not fully executed
RTE/Device/ATSAMV71Q21/startup_S...	
Reset_Handler	100% of 4 instructions
NMI_Handler	0% of 1 instructions
HardFault_Handler	0% of 1 instructions
MemManage_Handler	0% of 1 instructions
BusFault_Handler	0% of 1 instructions
UsageFault_Handler	0% of 1 instructions
SVC_Handler	0% of 1 instructions
DebugMon_Handler	0% of 1 instructions
PendSV_Handler	0% of 1 instructions
SysTick_Handler	0% of 1 instructions
Default_Handler	0% of 1 instructions
__user_initial_stackheap	100% of 5 instructions

## Saving Code Coverage information:

Code Coverage information is temporarily saved during a run and is displayed in various windows as already shown. It is possible to save this information in an ASCII file to create a report or for use in other programs.

**TIP:** To get help on Code Coverage, type Coverage in the Command window and press the F1 key.

You can Save Code Coverage in two formats:

1. In a binary file that can be later loaded back into µVision. Use the command Coverage Save *filename*.
2. In an ASCII file. You can either copy and paste from the Command window or use the log command:
  - 1) log > c:\cc\test.txt ; send CC data to this file. The specified directory must exist.
  - 2) coverage asm ; you can also specify a module or function.
  - 3) log off ; turn the log function off.

1) Here is a partial display using the command **coverage**. This displays and optionally saves everything.

```
\\Blinky\Blinky.c\SysTick_Handler - 100% (6 of 6 instructions executed)
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
  3 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\Blinky.c\Delay - 100% (9 of 9 instructions executed)
\\Blinky\system_MK60N512MD100.c\SystemInit - 100% (34 of 34 instructions executed)
  2 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\system_MK60N512MD100.c\SystemCoreClockUpdate - 31% (36 of 116 instructions executed)
  5 condjump(s) or IT-bcock(s) not fully executed
\\Blinky\startup_MK60N512MD100.s\__asm_0x27C - 47% (9 of 19 instructions executed)
\\Blinky\startup_MK60N512MD100.s\Reset_Handler - 100% (4 of 4 instructions executed)
\\Blinky\startup_MK60N512MD100.s\NMI_Handler - 0% (0 of 1 instructions executed)
\\Blinky\startup_MK60N512MD100.s\HardFault_Handler - 0% (0 of 1 instructions executed)
\\Blinky\startup_MK60N512MD100.s\MemManage_Handler - 0% (0 of 1 instructions executed)
```

2) The command **coverage asm** produces this listing (partial is shown):

```
\\Blinky\Blinky.c\SysTick_Handler - 100% (6 of 6 instructions executed)
EX | 0x000002B8 SysTick_Handler:
EX | 0x000002B8 483D LDR r0,[pc,#244] ; @0x000003B0
EX | 0x000002BA 6800 LDR r0,[r0,#0x00]
EX | 0x000002BC 1C40 ADDS r0,r0,#1
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
  3 condjump(s) or IT-bcock(s) not fully executed
EX | 0x000002C4 main:
EX | 0x000002C4 F04F34FF MOV r4,#0xFFFFFFFF
EX | 0x000002C8 2501 MOVS r5,#0x01
EX | 0x000002CA F000F8CB BL.W SystemCoreClockUpdate (0x00000464)
```

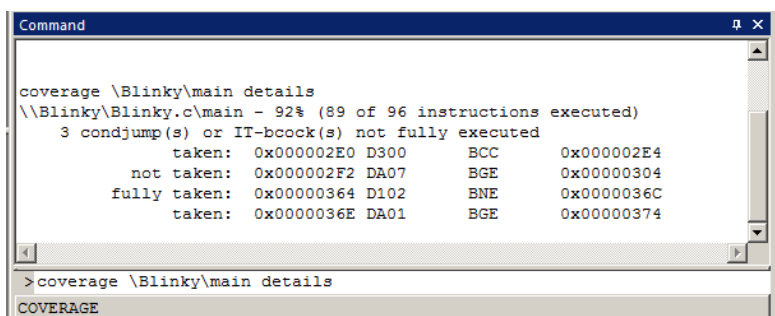
The first column above describes the execution as follows:

<b>NE</b>	Not Executed
<b>FT</b>	Branch is fully taken
<b>NT</b>	Branch is not taken
<b>AT</b>	Branch is always taken.
<b>EX</b>	Instruction was executed (at least once)

3) Shown here is an example using:  
**coverage \Blinky\main details**

If the log command is run, this will be saved/appended to the specified file.

You can enter the command **coverage** with various options to see what is displayed.





```
Command
coverage \Blinky\main details
\\Blinky\Blinky.c\main - 92% (89 of 96 instructions executed)
  3 condjump(s) or IT-bcock(s) not fully executed
    taken: 0x000002E0 D300 BCC 0x000002E4
    not taken: 0x000002F2 DA07 BGE 0x00000304
    fully taken: 0x00000364 D102 BNE 0x0000036C
    taken: 0x0000036E DA01 BGE 0x00000374
>coverage \Blinky\main details
COVERAGE
```

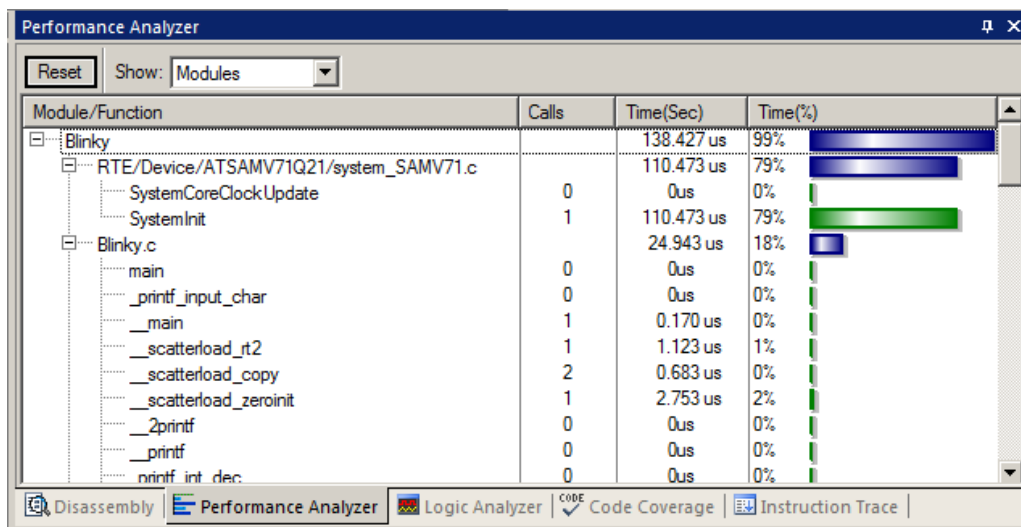



## 26) Performance Analysis (PA):

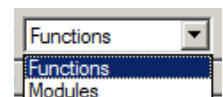
Performance Analysis tells you how much time was spent in each function. It is useful to optimize your code for speed.




Keil provides Performance Analysis with the  $\mu$ Vision simulator or with ETM and the ULINKpro. The number of total calls made as well as the total time spent in each function is displayed. A graphical display is generated for a quick reference. If you are optimizing for speed, work first on those functions taking the longest time to execute.

1. Use the same setup as used with Code Coverage. Do not have the program running.
2. Select View/Analysis Windows/Performance Analysis. A window similar to the one below will open up.
3. Click on RESET . This resets the SAMV7 processor and keeps it at RESET,
4. Click on the RESET  in the Performance Analysis (PA) window as shown below. This clears the PA window.
5. Enter g,main in the Command window to run the program to the beginning of main() in Blinky.c.
6. Do **not** click on RUN yet Expand some of the module names as shown below.
7. Times and number of calls has been collected in this short run from RESET to main().



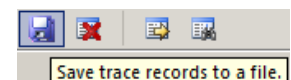
8. Click on the RUN icon. 
9. Note the display changes in real-time while the program Blinky is running. There is no need to stop the processor to collect the information. No code stubs are needed in your source files. Most time is spent in the Delay function.
10. Select Functions from the pull down box as shown here and notice the difference.



11. Click on the PA RESET icon.  Watch as new data is displayed in the PA window.
12. When you are done, STOP the program  and exit Debug mode. 

**TIP:** The Performance Analyzer uses ETM to collect its raw data.

**TIP:** You can save the Trace data to a file using Save Trace icon in the Trace Data window:



## 27) Execution Profiling:

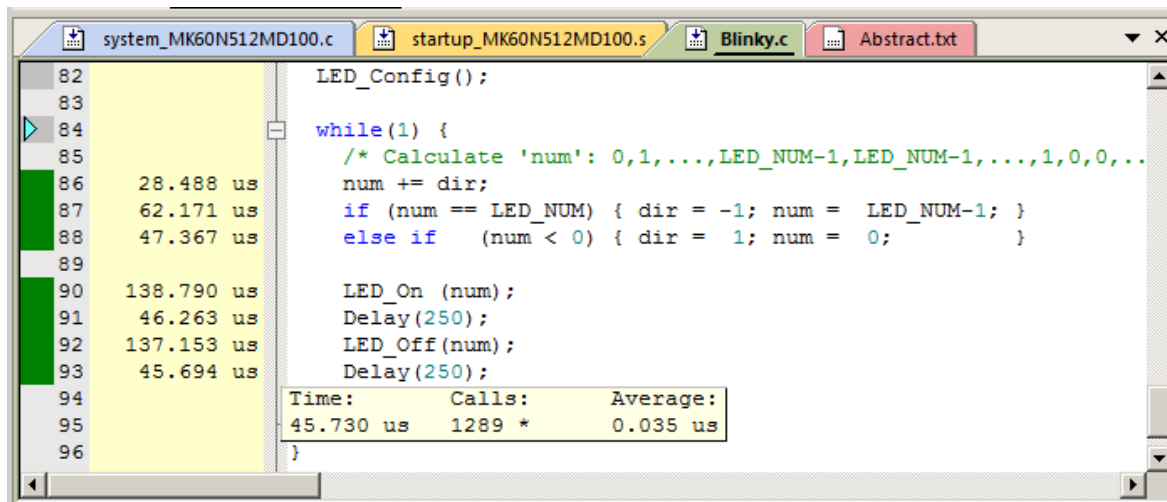
Execution profiling is used to display how much time a C source line took to execute and how many times it was called. This information is provided by the ETM trace in real time while the program keeps running.

The µVision simulator also provides Execution Profiling.


1. Enter Debug mode.
2. Select Debug/Execution Profiling/Show Time.
3. Click on RUN.
4. In the left margin of the disassembly and C source windows will display various time values.
5. The times will start to fill up as shown below right:
6. Click inside the yellow margin of Blinky.c to refresh it.
7. This is done in real-time and without stealing CPU cycles.
8. Hover the cursor over a time and ands more information appears as in the yellow box here:



Time:	Calls:	Average:
19.599 s	139910257 *	0.140 µs

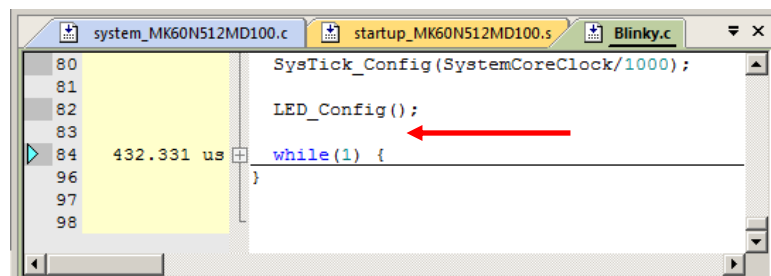
9. Recall you can also select Show Calls and this information rather than the execution times will be displayed in the left margin.



## Outlining:

Each place there is a small square with a “-” sign  can be collapsed down to compress the associated source files together.

- 1) Click in the square near the while(1) loop near line 84 as shown here:
- 2) Note the section you blocked is now collapsed and the times are added together where the red arrow points.
- 3) Click on the + to expand it.
- 4) Stop the program .
- 5) Exit Debug mode .



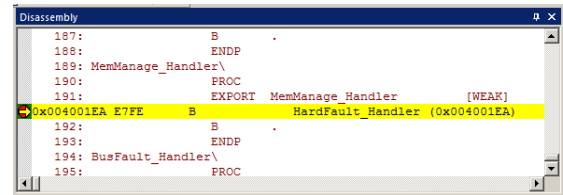
## 28) In-the-Weeds Example:

Some of the hardest problems to solve are those when a crash has occurred and you have no clue what caused this – you only know that it happened and the stack is corrupted or provides no useful clues. Modern programs tend to be asynchronous with interrupts and RTOS task switching plus unexpected and spurious events. Having a recording of the program flow is useful especially when a problem occurs and the consequences are not immediately visible. Another problem is detecting race conditions and determining how to fix them. ETM trace handles these problems and others easily and it is not hard to use.

If a Bus Fault occurs in our example, the CPU will end up at 0x0040 01EA as shown in the disassembly window below. This is the Bus Fault handler. This is a branch to itself and will run this Branch instruction forever. The trace buffer will save millions of the same branch instructions. This is not very useful.

This exception vector is found in the file startup\_SAMV71.s. If we set a breakpoint by clicking on the Hard Fault handler and run the program: at the next Bus Fault event the CPU will jump to this location.

The breakpoint will stop the CPU and also the trace collection. The trace buffer will be visible and extremely useful to investigate and determine the cause of the crash.



1. Use the Blinky example from the previous exercise, enter Debug mode.
2. Locate the Hard fault vector near address 0x0040 01EA in the Disassembly window as shown above or near line 187 startup\_SAMV71.s. You are looking for a B instruction with the label HardFault\_Handler as shown above.
3. Set a breakpoint at this point. A red circle will appear.
4. Run the Blinky example for a few seconds and click on STOP. Single Step to enter either Led\_On or Led\_Off function. You can see this in the Call Stack and Locals window.

**TIP:** You need to enter any function that will process a few instructions and then return: We will put a bogus value (00) into the LR (Link Register) to create a Hard Fault on the next function return.

5. Clear the Trace Data window by clicking on the Clear Trace icon: This is to help clarify what is happening.
6. In the Register window, double-click on R14 (LR) register and set it to zero. LR contains the return address from a subroutine. This is guaranteed to cause a hard fault when the processor tries to execute an instruction at the initial SP location in Flash at 0x00. This will cause a real problem since this is not a real instruction: it contains an address in RAM which is the location of the initial Stack Pointer at RESET.
7. Click on RUN and almost immediately the program will stop on the Hard Fault exception branch always instruction.
8. In the Trace Data window you will find the remainder of the LED function with the BX LR at the end.
9. The B instruction at the Hard Fault vector was not executed because ARM CoreSight hardware breakpoints do not execute the instruction they are set to when they stop the program. They are no-skid breakpoints.

Time	Address / Port	Instruction / Data	Src Code / Trigger Addr	Function
	X: 0x0040036C	MOV r1,r0	int32_t LED_On (uint32_t num) {	LED_On
	X: 0x0040036E	CMP r1,#0x02	if (num < NUM_LEDS) {	LED_On
3.056 209 233 s	X: 0x00400370	*BCS 0x0040038A		LED_On
	X: 0x00400372	*CBNZ r1,0x00400380	if (num == 0)	LED_On
	X: 0x00400374	LDR r0,[pc,#24] ; @0x00400390	PIOA->PIO_CODR = led_mask[num];	LED_On
	X: 0x00400376	LDR r0,[r0,r1,LSL #2]		LED_On
	X: 0x0040037A	LDR r2,[pc,#24] ; @0x00400394		LED_On
	X: 0x0040037C	STR r0,[r2,#0x00]		LED_On
3.056 210 033 s	X: 0x0040037E	B 0x0040038A		LED_On
	X: 0x0040038A	MOVS r0,#0x00	return 0;	LED_On
	X: 0x0040038C	BX lr		LED_On
3.056 211 160 s	X: 0x004001EA	B HardFault_Handler (0x004001EA)	B	HardFault_Handler

10. Single Step to run a B at the HardFault Handler which will be displayed at the end of the Trace as shown above. You can see this is the HardFault handler. This is also listed in the Function column.

This is admittedly a contrived example but it clearly shows how quickly and easily ETM trace can show you program flow problems.

These types of problems are very hard to solve and usually take a long time.

## 29) Serial Wire Viewer and ETM Trace Summary:

### Serial Wire Viewer can see:

- Global variables.
- Static variables.
- Structures.
- Peripheral registers – just read or write to them.
- Can't see local variables. (just make them global or static).
- Can't see DMA transfers – DMA bypasses CPU and SWV by definition.

### Serial Wire Viewer displays in various ways:

- PC Samples.
- Data reads and writes.
- Exception and interrupt events.
- CPU counters.
- Timestamps.

### These are the types of problems that can be found with a quality ETM trace:

- ETM facilitates Code Coverage, Performance Analysis and program flow debugging and analysis.
- Trace adds significant power to debugging efforts. Tells where the program has been.
- A recorded history of the program execution *in the order it happened*.
- Trace can often find nasty problems very quickly. Weeks or months can be replaced by minutes.
- Especially where the bug occurs a long time before the consequences are seen.
- Or where the state of the system disappears with a change in scope(s).
- Pointer problems. Illegal instructions and data aborts (such as misaligned writes).
- Code overwrites – writes to Flash, unexpected writes to peripheral registers (SFRs), a corrupted stack.  
*How did I get here ?*
- Out of bounds data. Uninitialized variables and arrays.
- Stack overflows. What causes the stack to grow bigger than it should ?
- Runaway programs: your program has gone off into the weeds and you need to know what instruction caused this. Is very tough to find these problems without a trace. ETM trace is best for this. Especially where the Stack is corrupted or destroyed and unreliable.
- Communication protocol and timing issues. System timing problems.

### 30) Document Resources:

See [www.keil.com/Atmel](http://www.keil.com/Atmel)

#### Books:

1. **NEW!** **Getting Started with MDK 5:** Obtain this free book here: [www.keil.com/mdk5/](http://www.keil.com/mdk5/)
2. There is a good selection of books available on ARM: [www.arm.com/support/resources/arm-books/index.php](http://www.arm.com/support/resources/arm-books/index.php)
3.  $\mu$ Vision contains a window titled Books. Many documents including data sheets are located there.
4. **A list of resources is located at:** [www.arm.com/products/processors/cortex-m/index.php](http://www.arm.com/products/processors/cortex-m/index.php)  
Click on the Resources tab. Or select the Cortex-M processor you want in the Processor panel on the left.
5. Or search for the Cortex-M processor you want on [www.arm.com](http://www.arm.com).
6. **The Definitive Guide to the ARM Cortex-M0/M0+** by Joseph Yiu. Search the web for retailers.
7. **The Definitive Guide to the ARM Cortex-M3/M4** by Joseph Yiu. Search the web for retailers.
8. **Embedded Systems: Introduction to Arm Cortex-M Microcontrollers** (3 volumes) by Jonathan Valvano
9. MOOC: Massive Open Online Class: University of Texas: <http://users.ece.utexas.edu/~valvano/>

#### Application Notes:

10. **NEW!** ARM Compiler Qualification Kit: Compiler Safety Certification: [www.keil.com/safety](http://www.keil.com/safety)
11. Using Cortex-M3 and Cortex-M4 Fault Exceptions [www.keil.com/appnotes/files/apnt209.pdf](http://www.keil.com/appnotes/files/apnt209.pdf)
12. CAN Primer: [www.keil.com/appnotes/files/apnt\\_247.pdf](http://www.keil.com/appnotes/files/apnt_247.pdf)
13. Segger emWin GUIBuilder with  $\mu$ Vision™ [www.keil.com/appnotes/files/apnt\\_234.pdf](http://www.keil.com/appnotes/files/apnt_234.pdf)
14. Porting mbed Project to Keil MDK™ [www.keil.com/appnotes/docs/apnt\\_207.asp](http://www.keil.com/appnotes/docs/apnt_207.asp)
15. MDK-ARM™ Compiler Optimizations [www.keil.com/appnotes/docs/apnt\\_202.asp](http://www.keil.com/appnotes/docs/apnt_202.asp)
16. GNU tools (GCC) for use with  $\mu$ Vision <https://launchpad.net/gcc-arm-embedded>
17. RTX CMSIS-RTOS Download [www.keil.com/demo/eval/rtx.htm](http://www.keil.com/demo/eval/rtx.htm)
18. Barrier Instructions <http://infocenter.arm.com/help/topic/com.arm.doc.dai0321a/index.html>
19. Lazy Stacking on the Cortex-M4: [www.arm.com](http://www.arm.com) and search for DAI0298A
20. Cortex Debug Connectors: [www.arm.com](http://www.arm.com) and search for cortex\_debug\_connectors.pdf *or*  
[www.keil.com/coresight/coresight-connectors](http://www.keil.com/coresight/coresight-connectors)
21. Sending ITM printf to external Windows applications: [http://www.keil.com/appnotes/docs/apnt\\_240.asp](http://www.keil.com/appnotes/docs/apnt_240.asp)
22. FlexMemory configuration using MDK [www.keil.com/appnotes/files/apnt220.pdf](http://www.keil.com/appnotes/files/apnt220.pdf)
23. Sending ITM printf to external Windows applications: [www.keil.com/appnotes/docs/apnt\\_240.asp](http://www.keil.com/appnotes/docs/apnt_240.asp)
24. **NEW!** Migrating Cortex-M3/M4 to Cortex-M7 processors: [www.keil.com/appnotes/docs/apnt\\_270.asp](http://www.keil.com/appnotes/docs/apnt_270.asp)

**Community Forums:** [www.keil.com/forum](http://www.keil.com/forum) and <http://community.arm.com/groups/tools/content>

**ARM University program:** [www.arm.com/university](http://www.arm.com/university). Email: [university@arm.com](mailto:university@arm.com)

**mbed™:** <http://mbed.org> **CMSIS:** [www.arm.com/cmsis](http://www.arm.com/cmsis).

For comments or corrections on this document please email [bob.boys@arm.com](mailto:bob.boys@arm.com).



### 31) Keil Products and Contact Information: See [www.keil.com/Atmel](http://www.keil.com/Atmel)

#### Keil Microcontroller Development Kit (MDK-ARM™)

- MDK-Lite (Evaluation version) \$0
- MDK-Atmel™ For all Atmel Cortex-M processors. 1 year term license
- MDK-ARM-CM™ For all Cortex-M series processors only – unlimited code limit
- MDK-Standard (unlimited compile and debug code and data size)
- MDK-Professional (Includes Flash File, TCP/IP, USB and Graphics driver libraries for most processors)



#### USB-JTAG adapter (for Flash programming too)

- ULINK2 - (ULINK2 and ME - SWV only – no ETM)
  - ULINK-ME – sold only with a board by Keil or OEM.
  - ULINKpro – Cortex-Mx SWV & ETM trace. Very fast Flash programming.
  - ULINKpro D – same as ULINKpro but without ETM trace.
  - MDK also supports Atmel SAM-ICE, Atmel-ICE, Atmel EDBG, CMSIS-DAP and Segger J-Link Debug adapters.
- **For special promotional or quantity pricing and offers, please contact Keil Sales. In USA, 1-800-348-8051. For rest of the world: +49 89/456040-20 or [www.keil.com/distis/](http://www.keil.com/distis/)**



Keil RTX RTOS is now provided under a Berkeley BSD type license. This makes it free.

All versions, including MDK-Lite, include Keil RTX RTOS *with source code* !

Keil provides free DSP libraries for the Cortex-M0, Cortex-M3 Cortex-M4 and Cortex-M7.

Call Keil Sales for details on current pricing, specials and quantity discounts.

Sales can also provide advice about the various tools options available to you.

They will help you find various labs and appnotes that are useful.

All products are available from stock.

All products include Technical Support for 1 year. This is easily renewed.

Call Keil Sales for special university pricing. Go to [www.arm.com/university](http://www.arm.com/university) to view various programs and resources.

Keil supports many other Atmel processors including ARM9™ and 8051. See [www.keil.com/Atmel](http://www.keil.com/Atmel) for the complete list of Atmel support.

For Linux, Android and bare metal (no OS) support on Atmel Cortex-A processors such as SAMA5D3, please see DS-5 [www.arm.com/ds5](http://www.arm.com/ds5).



#### For more information:

**Keil Sales** In USA: [sales.us@keil.com](mailto:sales.us@keil.com) or 1-800-348-8051. Outside the US: [sales.intl@keil.com](mailto:sales.intl@keil.com) or +49 89/456040-20

**Keil Technical Support** in USA: [support.us@keil.com](mailto:support.us@keil.com) or 1-800-348-8051. Outside the US: [support.intl@keil.com](mailto:support.intl@keil.com).

**Your Local distributor:** [www.keil.com/distis/](http://www.keil.com/distis/)

For comments or corrections please email [bob.boys@arm.com](mailto:bob.boys@arm.com).

For more Atmel specific information, visit [www.keil.com/Atmel](http://www.keil.com/Atmel).

CMSIS Version 4: [www.arm.com/cmsis](http://www.arm.com/cmsis)

